Universidad
Carlos III de Madrid

Protocolo de comunicación Bluetooth y control por voz para PRESSMATIC

Trabajo de Fin de Grado

Autor: Alonso Rosado García

Tutor: Alberto Jardón Huete

Titulación: Grado en Tecnologías Industriales

Fecha: Septiembre 2014

Agradecimientos

Quisiera dar las gracias a todas aquellas personas que invierten sus esfuerzos en intentar mejorar el mundo y a sí mismos. Toda aportación, hasta la que pueda parecer más insignificante, es la que nos llevara a todos a un futuro mejor.

Resumen

En la actualidad, afortunadamente, existe un creciente foco de atención sobre las necesidades de aquellas personas que, debido a una gran diversidad de causas, ven en riesgo o pierden su autonomía personal. Es cierto sin embargo, que todavía queda mucho trabajo por hacer, y que lo hecho siempre se puede mejorar. Esto no quiere decir, ni mucho menos, que no existan personas trabajando duramente para ofrecer una vida mejor a aquellos que les rodean, y un ejemplo de ello es el proyecto PRESSMATIC ideado por Gabriel Barroso.

PRESSMATIC pretende ofrecer una solución automatizada y asequible al problema que presenta para algunas personas el movimiento de pinzado acometido por los dedos índice y pulgar. A fin de contribuir a mejorar el dispositivo, se pensó en la flexibilidad y movilidad que proporcionan las tecnologías inalámbricas. Es común encontrar estas tecnologías en otros dispositivos de asistencia dado que ofrecen grandes posibilidades de control y evitan el engorroso uso de cables.

Así pues, dentro de las tecnologías inalámbricas, se escogió la tecnología inalámbrica Bluetooth, pilar en el cual se basa este proyecto. La tecnología Bluetooth permite realizar conexiones seguras, fiables y se halla integrada en una gran mayoría de los dispositivos móviles disponibles actualmente e integrado en muchos de los dispositivos más básicos.

Por ello, a lo largo de este proyecto se crea un protocolo de comunicación Bluetooth para PRESSMATIC basado en emulación de puerto serie. Además también se elige un módulo Bluetooth asequible para integrar en PRESSMATIC y se configura para ajustarse a las necesidades del dispositivo.

También se desarrolla, aprovechando la capacidad de procesado y potencia del sistema operativo Android, una aplicación móvil de control de PRESSMATIC en colaboración con Alejandro Vega, de la cual se explica en detalle como programar la conexión Bluetooth. Además, como contribución añadida a esa aplicación y al control integrado en pantalla táctil para PRESSMATIC, proyecto realizado por Rodrigo Martín, se crear un conjunto de iconos gráficos para los interfaces de usuario.

Por último se aprovecha la conexión Bluetooth para crear un control por voz para PRESSMATIC basado en el módulo EasyVR.

Abstract

In the present times, as it is most fortunate, the focus over the needs of people who, due to a wide variety of causes, have lost or have risk of loosing their personal autonomy is increasing. It is true nonetheless that there is still plenty of work to do in this field and there is always place for improvement. By any means does this mean that there is not plenty of people working throughly to offer people around them a better living. An example of this is the PRESSMATIC project, concieved by Gabriel Barroso.

PRESSMATIC's objective is to offer an affordable automated solution to the challenge that the gripping movement produced with the index and thumb fingers may be. In pursue of the improvement of the device, the wireless technologies were approached due to their flexibility and movility. Nowadays it is posible to find these technologies among many other assistance devices offering wide control possibilities alongside avoiding the rather bothersome use of wires.

Among these technologies the Bluetooth Wireless Technology was chosen, and it is the foundation of this project. Bluetooth tecnollogy allows for secure and reliable connections, being part of most of the mobile devices actually available in the market and of many basic devices.

Thus, in this project, a Bluetooth communication protocol is created for PRESSMATIC based upon a serial port emulation. Also an affordable Bluetooth device is chosen for its integration within PRESSMATIC and is configured to best suit the devices needs.

Taking advantage of the power and processing capabilities of the Android operative system, a mobile PRESSMATIC control application is developed alongside Alejandro Vega, explaining with great detail how the Bluetooth connection is programmed and established.

Also, in addition to the contribution in the creation of the mobile application and also contributing to the integrated touchscreen control, several icons were created for the user interfaces.

Finally, using the Bluetooth connection, a voice recongnition control device is created based upon the EasyVR module.

Índice General

Índice de Figuras

Índice de Tablas

Capitulo 1

1. Introducción

1.1 Motivación del Proyecto

Este proyecto surge inicialmente en base al trabajo de desarrollo del ingeniero Gabriel Barroso de María. En su tesis "Dispositivo automático de apoyo para uso de herramientas requeridas de movimiento manual de pinzado" expone como, en la actualidad, millones de personas en todo el mundo sufren de algún tipo de diversidad funcional [1].

De entre estas personas, aquellas que han perdido cierta gracilidad manual no disponen de ningún dispositivo dedicado exclusivamente a hacer frente a ese problema. Así pues era necesario que el dispositivo cubriera tareas tan básicas y necesarias como agarrar objetos y utilizar herramientas de corte como tijeras o cortauñas, que pueden ser dificultadas e incluso impedidas por esta falta de destreza manual.

En su estudio del estado del arte, Gabriel evaluó el mercado y descubrió que la necesidad de dicho dispositivo no estaba cubierta. Así pues se procedió a la creación y desarrollo del concepto de PRESSMATIC [2], finalmente alcanzando el siguiente diseño.

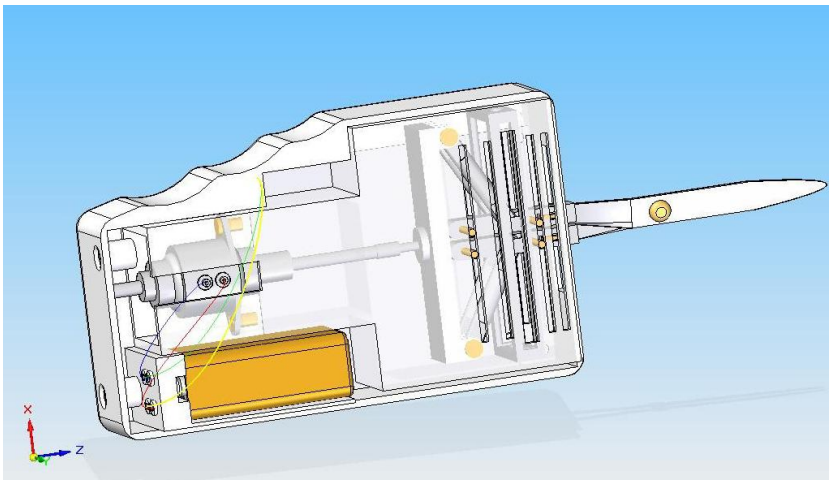

Figura 1.1 Concepto Inicial de PRESSMATIC

El concepto para la tesis de fin de máster fue el de un dispositivo electromecánico automático que dispondría de un cuerpo principal más un conjunto de cabezales intercambiables con distintas herramientas. El cuerpo principal debía ser ligero y ergonómico, manejable utilizando una sola mano e incorporar una pantalla táctil integrada que permitiera el control del dispositivo.

Tras la adjudicación de éste proyecto entre más de 400 propuestas presentadas a la Fundación Universia [2] se decide llevar adelante el concepto y convertirlo en una realidad. Es por ello por lo que se ofrecen diversos Trabajos de Fin de Grado, cada uno enfocado cumplir uno de los objetivos del proyecto PRESSMATIC.

Así pues, con el objetivo de cumplir con las necesidades de interacción con el usuario surgieron tres proyectos diferentes, entre ellos el presente proyecto, todos ellos dedicados a ofrecer diversas posibilidades de control sobre el dispositivo.

Estos tres proyectos, todos ellos dirigidos por el IP del proyecto Universia, son:

- Protocolo de comunicación Bluetooth y control por voz para PRESSMATIC, el presente proyecto [3].
- Programación en Android accesible para el control de PRESSMATIC, de Alejandro Vega Gómez [4].
- Programación e integración de interfaz de pantalla táctil accesible para PRESSMATIC, de Rodrigo Martín López [5].

Lógicamente estos proyectos tendrán una gran interrelación y varias partes comunes, por lo que en el presente documento se tratará remarcar las principales aportaciones al proyecto común PRESSMATIC

La integración de Bluetooth en el dispositivo, el objetivo de este proyecto, no es una forma de control de por sí, pero abre grandes posibilidades y opciones de interacción con PRESSMATIC. Un ejemplo de ello es el proyecto de desarrollo de una aplicación móvil en Android, otro de los proyectos relacionados con PRESSMATIC. La aplicación se valdría de la conexión creada a través del Bluetooth para ofrecer las capacidades de personalización y control de Android para manejar el dispositivo y así sacar partido de la funcionalidad de un Smartphone o Tablet.

Esto es sólo el principio, pues una conexión Bluetooth ofrece la posibilidad de crear dispositivos y aplicaciones al margen del cuerpo principal de PRESSMATIC para llegar a satisfacer las necesidades de todo tipo de usuarios que requieran de esas capacidades de adaptación y personalización.

Estos proyectos, a fin de ofrecer una experiencia unificada para el usuario final, estuvieron íntimamente interrelacionados y es por ello por lo que se pueden hallar contribuciones conjuntas y similitudes a lo largo de los tres proyectos que serán debidamente especificadas.

1.2 Objetivo del Proyecto

Añadir conectividad Bluetooth a un dispositivo siempre abre grandes posibilidades ya que mediante el uso de este protocolo se eliminan muchas incompatibilidades de comunicación entre distintos tipos de hardware y se sobrepasan muchas barreras y limitaciones de espacio y peso al externalizar los servicios.

Para el proyecto PRESSMATIC, el incorporar comunicación por Bluetooth significa mucho más si cabe. Las necesidades de cada persona pueden variar enormemente, aunque todas esas necesidades tengan como un punto común de partida el uso del dispositivo PRESSMATIC. El poder adaptar este dispositivo a las diferentes necesidades sin necesidad de modificar el hardware inicial resulta beneficioso tanto para desarrolladores como para usuarios, pero repercute positivamente sobre todo en el usuario final.

Así pues el objetivo de este trabajo es, ante todo, **crear un protocolo de control sencillo de implementar y robusto**, que permita un control inalámbrico y fiable sobre el dispositivo desde cualquier tipo de terminal. Para ello, es necesaria la **búsqueda e implementación de un módulo Bluetooth interno** en el propio dispositivo, más las modificaciones de software necesarias para trabajar con este módulo, además de realizar una serie de **propuestas** que ejemplifiquen y justifiquen el porqué de éste añadido.

De esta forma, gracias a este protocolo, el dispositivo PRESSMATIC se podrá **controlar indistintamente y simultáneamente** desde su pantalla táctil, desde una aplicación Android y/o desde un módulo de reconocimiento de voz. Además, utilizando el módulo Bluetooth se ha de **implementar una comunicación inalámbrica** entre PRESSMATIC y cualquiera de sus controladores externos, excepto, claro está, la pantalla integrada.

1.3 Fases del Proyecto

El proyecto se estructura en una serie de fases que representan el proceso de desarrollo del mismo. Estas fases son las que, paso a paso, determinaron el resultado final del proyecto y el contenido de este proyecto. A continuación se muestra una breve descripción de cada fase.

Fase 1: Búsqueda de información y documentación. Esta fase engloba todo el proceso de análisis y recopilación de documentación sobre las diversas tecnologías utilizadas en este proyecto, justo con la búsqueda de los materiales necesarios para la ejecución del mismo

Fase 2: Preparación del entorno de trabajo. En esta fase se instala y configura el software de trabajo necesario para el desarrollo del proyecto.

Fase 3: Configuración del módulo Bluetooth y creación del Protocolo. A lo largo de esta fase, se configuraron los módulos Bluetooth y se creó el protocolo de comunicación, fundamento de las siguientes fases.

Fase 4: Implementación de la conexión Bluetooth para la aplicación Android PRESSMATIC. En colaboración con Alejandro Vega se procedió crear una conexión Bluetooth estable para la aplicación PRESSMATIC.

Fase 5: Creación de un módulo de control por voz para PRESSMATIC. En base al protocolo Bluetooth creado y al módulo de reconocimiento de voz EasyVR se crea un módulo externo de control de PRESSMATIC.

Fase 6: Creación de set de iconos PRESSMATIC. En forma de colaboración con Rodrigo Martín y Alejandro Vega, se crea bajo su dirección varios sets de iconos accesibles para la aplicación Android y para la interfaz de la pantalla integrada.

Fase 7: Pruebas y retoques. Se realiza una serie de pruebas para comprobar que las diversas partes del proyecto siguen el funcionamiento esperado. En los casos necesarios se aplicaron soluciones y retoques.

Fase 8: Documentación del proyecto. En esta fase se redacta la memoria donde se describen los resultados obtenidos y los procesos seguidos durante las fases previas.

Fase 9: Evaluación del dispositivo y aplicación Android con usuarios reales. Esta fase, en marcha en el momento de la redacción del presente documento, es la encargada de validar los resultados en usuarios potenciales mediante ensayos clínicos en el HNPT.

1.4 Estructura de la memoria

En esta sección se muestra un breve resumen de los contenidos de la memoria.

El Capítulo 2 corresponde al Estado del Arte. En él ser realiza un análisis extenso y se justifica el uso de las tecnologías escogidas para el desarrollo del proyecto, centrándose sobre todo en la tecnología Bluetooth y su arquitectura.

El núcleo de la memoria está conformado por los diversos capítulos que engloban el desarrollo y descripción del proyecto. El Capitulo 3 describe el protocolo de control creado y la configuración del módulo Bluetooth para cumplir con los objetivos de desarrollo del proyecto. El Capítulo 4 recoge las aplicaciones desarrolladas para PRESSMATIC en base al protocolo creado en el Capítulo 3 y muestra los detalles del proceso de creación de éstas. El Capítulo 5 engloba el proceso de creación y justificación de los iconos para los interfaces gráficos de PRESSMATIC. Por último, el Capítulo 6 recoge las pruebas realizadas y las correcciones propuestas.

En el Capítulo 7 se exponen las conclusiones obtenidas y se da pie a futuras líneas de desarrollo que puedan surgir en base a este proyecto o que lo amplíen y mejoren.

Finalmente, en el Capítulo 8, se exponen los costos del proyecto y el presupuesto resultante.

Además, se añaden al documento la bibliografía utilizada junto con un glosario de términos y Anexos referentes a la creación de los iconos del proyecto.

Capítulo 2

2. Estado del Arte

En este capítulo sobre el estado del arte se estudian las bases y se justifican las elecciones de las diferentes tecnologías utilizadas a lo largo de la realización del proyecto. El proyecto consta de un núcleo central que implementa un puerto serie de forma inalámbrica basado en la tecnología inalámbrica Bluetooth y que ha de interactuar de manera predecible y estable con diversas aplicaciones secundarias, como una aplicación móvil y un módulo de reconocimiento de voz.

Estas aplicaciones, tras un proceso de selección y descarte entre las diversas posibilidades, se desarrollaron en base al SO Android y el entorno de desarrollo Arduino.

La aplicación Android , desarrollada conjuntamente con Alejandro Vega [4], aprovecha las diversas opciones de accesibilidad que ofrece Android a las cuales se suma la creación de iconos accesibles basada en el programa de diseño vectorial Inkscape para crear una aplicación que cumpla lo mas fidedignamente posible las normas de accesibilidad.

El reconocimiento de voz, basado en los chipsets ATMega y el entorno de desarrollo Arduino, es fruto de la interacción con el módulo EasyVR de reconocimiento de voz, ofreciendo una nueva forma de control que libere carga a la destreza manual al usuario.

En los siguientes puntos se explicara en detalle el porqué de las elecciones realizadas y se expondrán diversos aspectos de las tecnologías utilizadas para respaldarlas.

2.1 Bluetooth

La comunicación Bluetooth está basada una tecnología de comunicación inalámbrica de corta distancia bastante común hoy en día. Está integrada en dispositivos de tecnología móvil, ordenadores y hasta en dispositivos médicos con el propósito de sustituir las conexiones mediante cables sin perder la alta seguridad y fiabilidad de éstas.

Es debido a este compromiso con fiabilidad en la comunicación que el uso del Bluetooth se ha extendido y consolidado a lo largo de los años hasta en las tareas más cotidianas y que resulta tan adecuado para el desarrollo proyecto.

El Bluetooth como tal es una especificación industrial para redes WPAN o Wireless Personal Area Network que permite una transmisión de datos mediante radiofrecuencia o RF en la banda de 2,4 GHz.

2.1.1 Historia

Los fundamentos de la tecnología Bluetooth tuvieron su origen en la compañía Ericsson Mobile Communications, gracias al ingenio de un equipo de desarrollo localizado en la ciudad sueca de Lund.

Con el objetivo de desarrollar una tecnología inalámbrica compacta, barata y eficiente energéticamente el equipo desarrolló el prototipo Multi-Communicator Link , presentado en 1997. Viendo grandes posibilidades en la creación de un estándar para comunicaciones inalámbricas, ese mismo año, contactaron con Intel que, tras una reunión realizada en la ciudad de Lund, cambió el nombre de la tecnología a Bluetooth, en honor al rey danés Harald Blatand, cuyo nombre traducido al Inglés es Harald Bluetooth.

El año siguiente Intel fomentó la creación de un grupo de interés especial o SIG, conformado por cinco compañías y adoptando oficialmente el nombre de Bluetooth. Hacia el fin de ese mismo año el grupo integraba a más de 400 miembros [6], [7].

Versión del Estándar	Fecha de Adopción	Notas sobre la versión
1.0 - 1.0a - 1.0b - 1.0b+CE - 1.1	Julio de 1999 a Febrero de 2001	Este grupo de versiones representan el progreso desde la primera versión del borrador estándar a la primera versión utilizable incorporada en el estándar IEEE 802.15.1 – 2002 [8]
1.2	5 de Noviembre de 2003	Añadido de diversas funcionalidades para un mejor soporte de la transmisión de voz
2.0+EDR	4 de Noviembre de 2004	El añadido del EDR (Enhanced Data Rate) incrementa la velocidad del flujo de datos a 3 Mbps
2.1+EDR	26 de Julio de 2007	Se añade la opción de emparejamiento simple seguro
3.0+HS	21 de Abril de 2009	Se añade un nuevo canal de transmisión para incrementar la velocidad del flujo de datos a 10Mbps
4.0	30 de Junio de 2010	Se incluyen en el estándar los Bluetooth Low Energy y ATT y GATT en lugar de EDR
4.1	3 de Diciembre de 2013	Actualización de aspectos arquitectónicos para incluir nuevas funcionalidades

Tabla 2.1 Especificaciones Bluetooth hasta la fecha.

Desde entonces se han ido lanzando diversos estándares que han ido incrementando las funcionalidades y la fiabilidad de este servicio.

2.1.2 Arquitectura

El objetivo de la arquitectura Bluetooth es, ante todo, la interoperabilidad entre aplicaciones. Para asegurar que este requisito se alcanza, las aplicaciones se ejecutan siempre sobre una pila de protocolos idénticos y con un hardware de base común.

El hardware específico para realizar una conexión incluye un módulo de radio y una antena además de un módulo Baseband. Como añadido a este hardware básico hay un software encargado de realizar y controlar la conexión entre dos dispositivos Bluetooth , integrado en un microprocesador dedicado conocido como Link Manager. Este procesador se comunicará con los LM de otros dispositivos siguiendo el Link Manager Protocol o LMP.

El software del LM es solo uno de los distintos protocolos que es posible encontrar en la pila de protocolos. Esta pila engloba todo aquel protocolo utilizado en la comunicación mediante estos módulos, que pueden ser tanto específicos Bluetooth, como el propio LM o el L2CAP, como protocolos no específicos como OBEX (Objects Exchange Protocol),UDP (User Datagram Protocol) o TCP (Transmission Control Protocol) [9].

2.1.2.1 Pila de Protocolos

En base al propio L2CAP se han diseñado la gran mayoría de estos protocolos. Esto es debido a que, gracias a la especificación es abierta se pueden desarrollar nuevos protocolos de aplicación en capas superiores.

El número de protocolos existentes que se pueden reutilizar en las capas más altas para diferentes finalidades se maximiza gracias a este planteamiento, lo que quiere decir que la torre de protocolos ha sido diseñada con un claro objetivo, el de poder ofrecer una gran variedad de oportunidades de desarrollo de servicios por parte de los fabricantes.

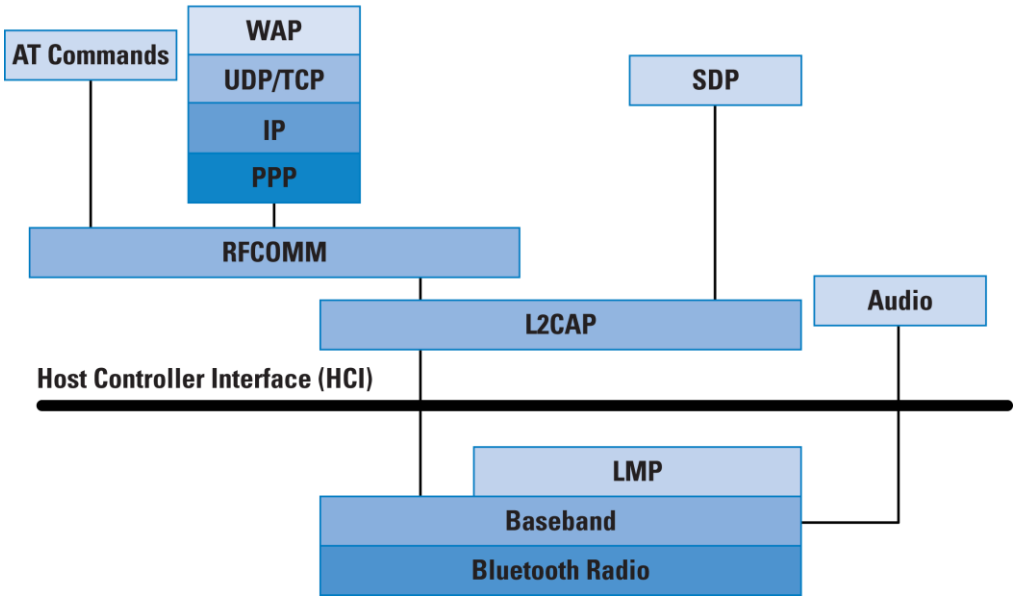

Figura 2.1 Pila de Protocolos Bluetooth

Así pues, las aplicaciones pueden trabajar utilizando una o varias de las columnas de protocolos especificadas por la torre en la siguiente figura. Algunas de ellas, como el SDP, son utilizadas únicamente como soportes de aplicaciones principales.

Siguiendo esta premisa se pueden dividir los protocolos Bluetooth en varias capas.

La primera de ella consta de los protocolos Bluetooth básicos o nucléicos (Bluetooth Core Protocols), que son el Baseband, el LMP, el L2CAP y el SDP. Estos protocolos fueron diseñados por el Bluetooth SIG y son utilizados por la gran mayoría de los dispositivos Bluetooth. También desarrollados por el SIG, los protocolos RFCOMM y TCS Binary fueron respectivamente basados en las especificaciones ETSI-TS 07.10 y la ITU-T Q.931

El LMP junto con el LM es el sistema que establece la conexión entre dispositivos, con ya se ha aclarado con anterioridad. El LMP consiste básicamente en una serie de PDUs (Protocol Data Units) que se envían entre dispositivos determinados por el AM_ADDR en la cabeza del paquete de datos. Los PDUs son siempre enviados como paquetes de un solo byte, por lo que el header también lo es.

A fin de establecer la conexión, el LM se encarga de todas las etapas del enlace, desde el establecimiento de la comunicación o enlace hasta la autentificación y configuración de éste.

Gracias al protocolo de gestión localiza a otros gestores y se comunica con ellos utilizando los servicios incluidos en el Link Controller.

Los principales servicios soportados o PDUs corresponden a:

- La Transmisión y Recepción de datos

- Emparejamiento, Autenticación, Encriptación y Conexión

- Establecimiento del modo de enlace y control de paquetes múltiples

El L2CAP [9] proporciona servicios de datos, tanto enfocados a la conexión como transmisiones sin conexión a las capas superiores de protocolos, con capacidades de multiplexación y de realizar operaciones de segmentación y recomposición o reensamblaje. El protocolo permite además que se transmitan paquetes de datos L2CAP de hasta 64 kilobytes de longitud a aplicaciones y protocolos de capas superiores.

El L2CAP se basa en la conexión Baseband, aunque solo soporta el tipo de conexión ACL (Asynchronous Connection-Less link). Para la comunicación se crean una serie de canales, entendiendo canal como una conexión lógica situada sobre la Baseband . Cada canal se asocia a un único protocolo al cual transmite los datos L2CAP recibidos y es referido por un identificador de canal CID. Se pueden asociar varios canales a la misma Baseband [3].

El SDP permite a las aplicaciones descubrir la existencia de servidores para las aplicaciones junto con sus atributos y propiedades. El servicio se basa en la comunicación entre un servidor SDP y un cliente SDP. El servidor es el que almacena la lista de registros de servicios, describiendo las características de los servicios ofrecidos. Un cliente SDP generalmente realizará una búsqueda de servicios acotada por esas características, aunque también puede realizar un browsing, buscando todos los servicios SDP disponibles

La segunda capa la engloban los llamados protocolos de reemplazo de cable o Cable Replacement Protocols, conformado únicamente por el RFCOMM.

El protocolo RFCOMM se basa en el protocolo L2CAP para crear una emulación de puertos serie. El protocolo creado resulta sencillo y con capacidad de soportar hasta 9 puertos serie, entendiendo estos como los estándar RS-232. A su vez, el protocolo llega a permitir hasta 60 canales simultáneos entre dos dispositivos Bluetooth, aunque el número de canales que se pueden utilizar simultáneamente depende de la implementación.

Una comunicación completa gestionada por RFCOMM involucrará a dos aplicaciones funcionando en dos dispositivos diferentes con un segmento de comunicación entre ellos.

Así pues el propósito de la RFCOMM es el de facilitar la creación de aplicaciones que utilizan puertos serie en los dispositivos en los que residen. El protocolo como tal sólo se encarga de las conexiones entre dispositivos mediante conexión directa más las conexiones entre el dispositivo y un módem para conexiones en red. Otras configuraciones son soportadas, como el Legacy Comm Device, encargado de hacer de puente entre un dispositivo Bluetooth y un dispositivo cableado.

La tercera capa es la conocida como los protocolos de control de Telefonía o Telephony Control Protocols, integrada por los protocolos TCS Binary y los AT-Commands.

Por último queda la capa de protocolos adaptados o Adapted Protocols, que engloban los restantes protocolos del diagrama extendido, como el UDP/TCP/IP, el OBEX o el WAP.

Las tres últimas capas son conocidas en su conjunto como los protocolos orientados a aplicación, que son abiertos y permiten a su vez la inclusión de nuevos protocolos, otorgando al estándar una gran flexibilidad.

2.1.2.2 Hardware básico de la comunicación Bluetooth

El hardware Bluetooth, como ya se ha explicado, es una sencilla conjunción de dos módulos: El módulo de radio y el módulo Baseband.

El **Módulo de radio** se encarga de la transmisión de la señal de manera modulada, trasladando los bits entre componentes de la conexión Bluetooth. La interfaz soporta un gran número de canales de radio y de niveles de frecuencia diferentes, además de diversos modos de modulación de alta fiabilidad.

Los niveles de frecuencia utilizados se hallan en la banda ISM, a 2,4GHz, dentro de la cual el protocolo Bluetooth realiza saltos de frecuencia unas 1600 veces por segundo. Ésta capacidad de operar en un mayor rango dinámico implica que, si una transmisión encuentra interferencias dentro de una frecuencia, los datos se enviarán posteriormente al cambiar de canal de transmisión. Los canales Bluetooth se encuentran a intervalos regulares de 1MHz desde los 2402 MHz a los 2480MHz, aunque en algunos países esta frecuencia de salto es más limitada. Además, para realizar una correcta comunicación, cada dispositivo tiene una identificación de 48 bits [10], [11].

En el estándar hay 2 modos de modulación. Un modo de implementación obligatoria basado en el GFSK y un modo de implementación opcional conocido como PSK, de mayor rapidez.

En el modo GFSK, la frecuencia de la onda portadora es desplazada para realizar la modulación, al corresponder el 1 a una desviación positiva y el 0 a una desviación negativa de la frecuencia. Esta señal modulada es filtrada utilizando un filtro con una respuesta Gaussiana curva, asegurando así que las bandas laterales no se extienden demasiado lejos de la portadora principal. Utilizando este método se consigue un ancho de banda de 1Mbps, que es el que corresponde al primer estándar Bluetooth.

El modo PSK fue el utilizado para otorgar al estándar 2.0 su EDR y consta de dos métodos diferentes, π/4 DQPSK y 8DPSK, basados en el uso de desfase de señal para transmitir la información, alcanzando un ancho de banda de 3 Mbps. Los siguientes estándares Bluetooth lograron incrementar este ancho de banda hasta los 25 Mbps no cambiando el tipo de modulación, sino trabajando conjuntamente con una capa física IEEE 802.11g.

Clase	Potencia Máxima (dBm)	Alcance
Clase 1	20 (Regulación de potencia obligatoria)	100 m
Clase 2	4	10 m
Clase 3	0	10 cm

Tabla 2.2 Clases de Módulos Bluetooth en función de su potencia y alcance

Además de la modulación del dispositivo, la potencia máxima también está estandarizada y englobada en tres clases. Aunque el Bluetooth está pensado para cortas distancias, puede llegar hasta los 100m de alcance.

El **módulo Baseband** se encarga de convertir el flujo de bits en tramas y definir algunos formatos clave. En su estándar más sencillo el maestro de cada red define ranuras de tiempo de 0.625 ms, correspondiendo las ranuras pares a las transmisiones del maestro y las impares a las de los esclavos, logrando así una multiplexación por tiempo. Cada trama puede estar conformada por 1,3 o 5 ranuras. Cada una de estas tramas se transmiten a través de un enlace entre el maestro y el esclavo, siendo de dos tipos: ACL, que, como ya se describió con anterioridad, está profundamente vinculada a la capa L2CAP, y el SCO, utilizado para datos en tiempo real.

En comparación con el SCO el ACL es de mucha menor complejidad, pudiendo enlazarse como máximo con un enlace entre esclavo y maestro, pero de menores garantías. Dado que el SCO está pensado para datos en tiempo real, como en el envío de audio en conexiones telefónicas, en lugar de usar retransmisiones utiliza una corrección de errores sin canal de retorno proporcionando alta fiabilidad. El SCO utiliza un máximo de 3 canales entre maestro y esclavo, de 64kpbs cada uno.

2.1.2.3 Redes Bluetooth

En base al hardware y software anteriormente descrito, se crean las redes de comunicación Bluetooth basadas en unidades llamadas piconets. Estas piconets están formadas por un nodo maestro y hasta un máximo de siete nodos esclavos activos. Para formar redes Bluetooth de mayor tamaño, varias piconets pueden interconectarse mediante el uso de nodos puente, formando un conjunto conocido como scatternet.

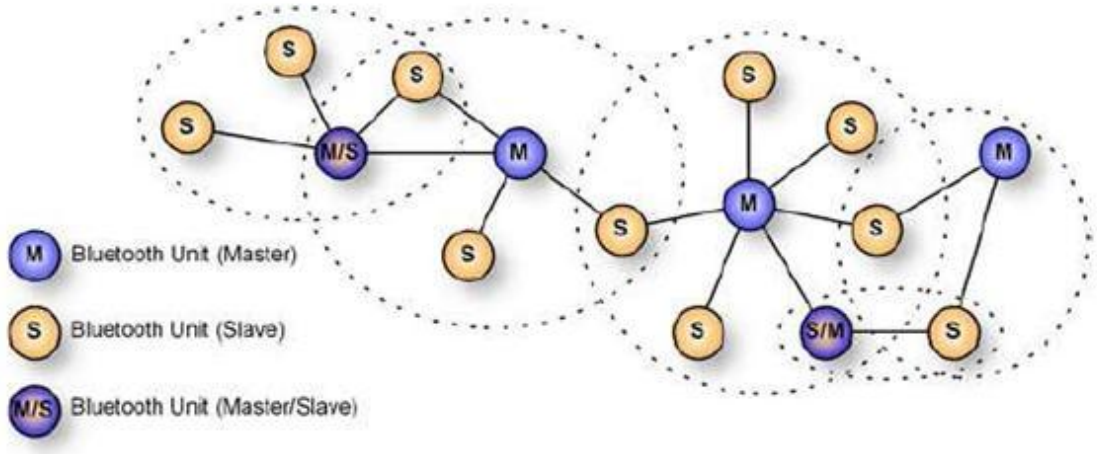

Figura 2.2 Ejemplo de una scatternet formada por varias piconets

Es importante aclarar el concepto de nodo activo, dado que el nodo maestro puede cambiar el estado de los nodos esclavo a bajo consumo a fin de ahorrar energía. A estos nodos se les conoce como nodos estacionados, y un maestro puede tener un total de 255 de estos nodos asignados, aunque sólo pueda tener 7 de esos nodos activos a la vez.

2.1.2.4 Seguridad

Uno de los principales objetivos de la creación del Bluetooth era ofrecer una alternativa inalámbrica segura y fiable. Así pues, resulta fundamental asegurar la integridad y privacidad de las transmisiones tal y como se harían a través de un cable. Es por ello que en los estándares Bluetooth se han asegurado de proporcionar cuatro modos con distintos niveles de seguridad de implementación obligatoria en cualquier módulo de comunicación.

Estos modos se fundamentan la aplicación de tres procesos: La autenticación, la autorización y la confidencialidad [12].

La autenticación es un proceso que verifica la identidad de los dispositivos de comunicación. También es posible implementar la verificación de usuario, aunque no forma parte del cuerpo principal de elementos de seguridad.

La autorización es sencillamente un proceso por el cual se habilita un permiso para usar el servicio de comunicación, impidiendo el acceso a aquellos dispositivos no autorizados.

Por último la confidencialidad se asegura de evitar el acceso a los datos a usuarios no autorizados.

En torno a estas bases se plantean:

El **modo 1** es el menos seguro de los métodos implementados, al abrir la conexión deshabilitando la autenticación y el cifrado. Básicamente, un dispositivo operando en este modo no emplea ningún mecanismo para evitar la conexión de un dispositivo Bluetooth no autorizado y es susceptible a brechas de seguridad. Es por ello que éste modo no es soportado por módulos que apliquen estándares a partir del core 2.0.

El **modo 2** inicia los procedimientos de seguridad tras el establecimiento de un canal entre los niveles LM y L2CAP. Un gestor de seguridad se encarga del control al acceso a servicios y dispositivos.

El **modo 3** inicia los procedimientos de seguridad antes de establecer ningún canal. Además de realizar un cifrado requiere de autenticación PIN y utiliza seguridad MAC. Se genera un emparejamiento o “pairing” cuando los dos dispositivos se comunican por primera vez, creando una clave de linkado secreta compartida entre ambos dispositivos.

A partir del core 2.1+EDR se incorpora el **modo 4**. En él los procedimientos de seguridad se inician después de la instalación del enlace. Se utiliza el método conocido como Secure Simple Pairing que se basa en técnicas de intercambio y generación de claves para el enlace, además de utilizar los algoritmos de encriptación ya incluidos en el core 2.0+EDR.

En este trabajo, para realizar el emparejado entre dos módulos Bluetooth HC-05 se utilizará el modo 2 de, mientras que para emparejarlo con el dispositivo Android se utilizará el modo 3.

2.1.2.5 Paquetes de Datos

Dentro de las redes descritas y una vez se ha asegurado la conexión, los datos se intercambian en forma de paquetes de datos. Un paquete de datos tiene un tamaño determinado principalmente por el número de ranuras que ocupa en su trama correspondiente. A pesar de esta variación en tamaño, la composición del paquete esta estandarizada [13].

El **código de acceso** corresponde a los primeros 72 bits enviados. Es utilizado en funciones de sincronización, identificación y compensación. Es, además, parte común de todo paquete dentro de una misma piconet para así poder acceder al canal.

La **cabecera** es la portadora de la información del control del enlace y corresponde a los 54 bits enviados a continuación del código de acceso. Contiene datos como el tipo de paquetes y el número de slots o ranuras que va a ocupar, el bit de control de flujo y la dirección AM_ADDR.

Si alguno de los datos de la cabecera o del código de acceso es incorrecto, el paquete recibido no es considerado válido y pasa a ser ignorado.

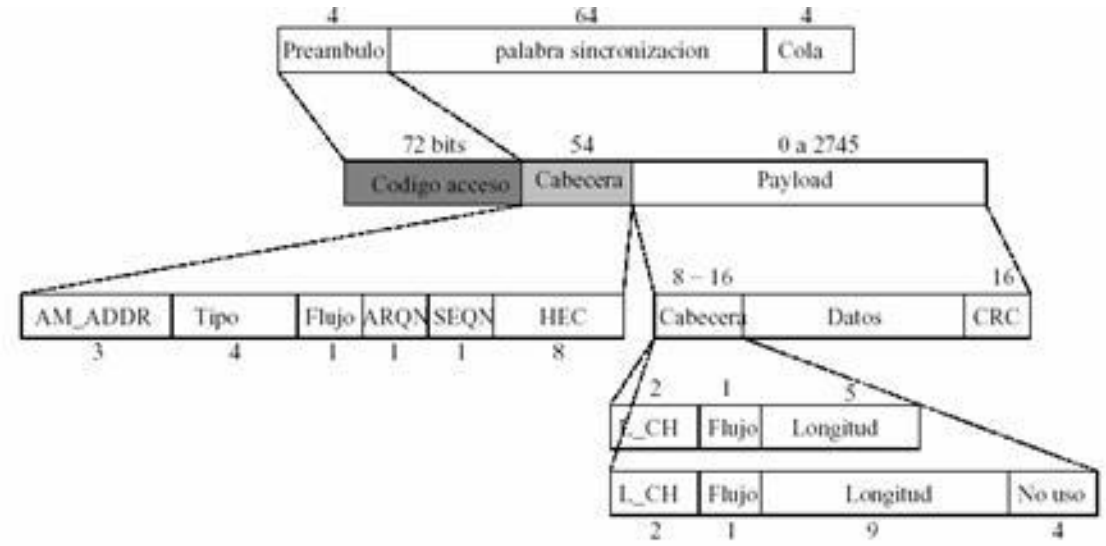

Figura 2.3 Formato de un paquete de datos Bluetooth

Por último el **campo de datos** ocupa los restantes bits de la transmisión llegando hasta 2746 bits, en función del número de slots utilizados. Estos datos son fundamentalmente la información útil a transmitir, y la comunicación resultará más eficiente cuanto mayor sea la cantidad de datos transmitida en este campo.

2.1.3 Elección del Módulo

En la actualidad hay cientos de módulos diferentes en el mercado implementando diversas versiones de las especificaciones Bluetooth. La más reciente divulgada por el grupo de interés especial (Special Interest Group o SIG) es la Core Version 4.1 de las Adopted Bluetooth Core Specifications. De esta especificación y de su hermana la 4.0 existen varios módulos comercializados a precios relativamente razonables y que cumplirían con creces las especificaciones

Si realizamos una consulta en diversos mercados online, podemos encontrar los siguientes módulos, ejemplo del tipo de módulos que cumplen esas normativas.

Módulo	Versión Bluetooth Core	Precio	fuente
ZBmodule CC2540	V4.0	14,30 US$	www.dx.com
Panasonic Pan2016	v4.0	11.70 €	www.mouser.es

Tabla 2.3 Módulos 4.0, comparación de precios

Sin embargo, dado que la carga de transferencia de datos requerida por el dispositivo resulta tan reducida que un módulo del protocolo 2.0 o 2.1 podría hacerlo, se sopesó la idea de utilizar una

especificación anterior. Dada la retrocompatibilidad de cores posteriores con las versiones anteriores de la especificación, esto significa que módulos basados en versiones posteriores del core podrán conectar con el dispositivo si son compatibles el mismo tipo de comunicación estándar, RFCOMM, SSP o TCS, todas basadas en L2CAP.

RFCOMM es la comunicación más intuitiva, al ser también conocida en ocasiones como emulación de puertos serie. Esto la convierte en la más sencilla de integrar en Arduino además de ser la adoptada por la gran mayoría de dispositivos Android, lo cual resulta de gran importancia más adelante. Es por ello por lo que se hizo una búsqueda centrada en estos parámetros.

El resultado más adecuado dentro del mercado fue el módulo HC-05

Módulo	Versión Bluetooth Core	Precio	Fuente
HC-05	v2.0	5.76 US$	www.dx.com

Tabla 2.4 Cualidades del Módulo HC-05

Su reducido precio junto con la capacidad de configurarlo tanto como maestro como esclavo mediante comandos AT, además de un alto grado de personalización y compatibilidad comprobada con Android, decantaron la elección del módulo a este dispositivo.

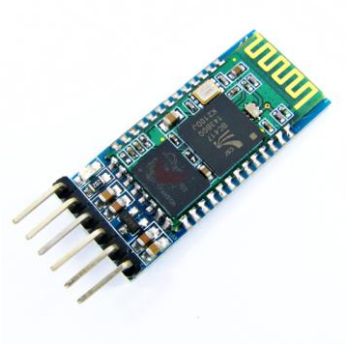

Figura 2.4 Módulo HC-05

Por último, como es posible apreciar en la figura 2.6, el módulo en si es de dimensiones bastante reducidas (35mm x 15mm x 3.5mm), lo cual lo hace idóneo para su integración en PRESSMATIC.

2.2 Plataforma de Desarrollo Arduino

La plataforma de desarrollo Arduino se ha convertido en una de las más populares de la actualidad. Basada en los microprocesadores ATMEGA y en el entorno de desarrollo sencillo proporcionado por sus creadores, las placas Arduino han servido de base de muchos proyectos y han generado una comunidad siempre creciente de desarrolladores.

Gran parte de éste éxito es el carácter Open Source o de software y hardware libre del proyecto, que además permite integrar los microchips ATMEGA y su programación realizada en Arduino en productos comerciales sin coste alguno.

Además, el desarrollo del hardware de control de PRESSMATIC se basa en un chipset ATMEGA programado en entorno de Arduino, lo cual, a fin de facilitar una colaboración entre los distintos proyectos, volvía a decantar la balanza a favor de la utilización de Arduino.

El lenguaje de este entorno, un híbrido simplificado entre los lenguajes de programación C y C++, resulta sencillo de aprender gracias a la experiencia previa adquirida a lo largo del Grado con ambos lenguajes [14].

El utilizar Arduino fue pues una decisión sencilla. Una gran comunidad de desarrolladores, sencillez de implementación y una curva de aprendizaje de no mucha pendiente decantaron la balanza hacia el desarrollo en esta plataforma de toda aquella aplicación o necesidad a lo largo del proyecto que requiriera un microprocesador subyacente.

Durante este proyecto se utilizaron dos módulos: el ya no comercializado Arduino UNO, retirado en favor del Arduino DUE, y el Arduino NANO.

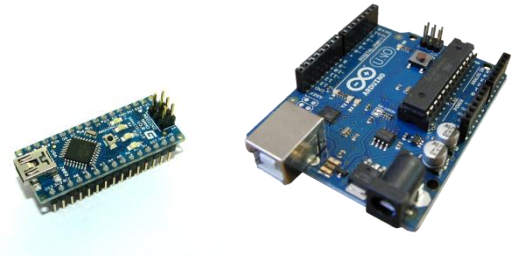

Figura 2.5 Arcuino NANO y Arduino UNO

Las primeras fases del proyecto utilizaron el Arduino UNO, para posteriormente pasar a utilizar Arduino NANO, de muy similares especificaciones aunque de menor tamaño. Todos los programas presentados en este proyecto son compatibles en ambos chipsets.

	Arduino UNO	**Arduino NANO**
Microcontrolador	ATmega328	ATmega328
Voltaje de operación	5V	5V
Velocidad de Reloj	16MHz	16MHz
Memoria Flash	32KB	32KB
Parejas de Pines Seriales	1	1

Tabla 2.5 Comparativa de Caracteristicas de Arduino UNO y Arduino NANO

2.3 Android como plataforma de desarrollo para la aplicación PRESSMATIC

La decisión de usar del sistema operativo Android fue tomada tras una investigación exhaustiva del Estado del Arte actual en cuanto a sistemas operativos y dispositivos móviles realizado por Alejandro Vega, principal desarrollador de la aplicación móvil PRESSMATIC.

Esta elección fue sustentada sobre todo por razones basadas tanto en el mercado actual de dispositivos móviles como en el propio entorno de desarrollo de aplicaciones Android.

En 2013, el SO Android de Google se perfilaba con el líder absoluto en el mercado de la tecnología móvil instalado en un 75% de dispositivos comercializados. Como parte de los objetivos de PRESSMATIC, alcanzar al mayor número de personas posible con ésta aplicación resulta muy atractivo, y el dominio de mercado de Android decanta eso en su favor [15].

Por otro lado, Google facilita su SDK en un módulo multiplataforma en el cual agrupa las herramientas necesarias para el desarrollo de cualquier aplicación. Esto implica que un desarrollador con conocimientos básicos de Java puede lanzarse a programar aplicaciones en Android en la plataforma de su elección [16].

Esto repercute en la comunidad de programadores que utilizan Android, que es mucho mayor que la de su más directo competidor, IOS de Apple, que al obligar a disponer de un ordenador Mac para poder programar su Smartphone IPhone, es más reducida.

Así pues, se desarrolló la aplicación para móviles Android, lo cual también tuvo repercusiones en el desarrollo del protocolo Bluetooth.

Figura 2.6 Logo del sistema operativo Android

2.4 Reconocimiento de Voz

El reconocimiento automático del habla es la capacidad de un sistema de procesar una señal de voz emitida por un ser humano a fin de reconocer el mensaje contenido en ésta y actuar en consecuencia.

Esta capacidad está integrada actualmente en casi todos los dispositivos electrónicos con una cierta potencia de procesado y tiene como objetivo proporcionar la posibilidad de interacción mediante comunicación hablada entre seres humanos y computadoras. Sin embargo, no está demasiado extendido su uso en general, en parte posiblemente a que el público mayoritario prefiere una interacción tradicional.

Sin embargo, muchas aplicaciones punteras de robótica e inteligencia artificial, domótica, accesibilidad e incluso militares integran el uso del reconocimiento de voz en la actualidad.

Un ejemplo de estas aplicaciones es el Robot Maggie, desarrollado en el laboratorio de Robótica de la universidad Carlos III de Madrid, cuyo objetivo es la interacción y comunicación con seres humanos de forma autónoma [17].

Figura 2.7 Robot Maggie, diseñado por ASROB en la Universidad Carlos III de Madrid

2.4.1 Historia

Los inicios del reconocimiento de voz se remontan a los años 50, cuando en los laboratorios Bell se diseñó el sistema "Audrey", capaz de reconocer dígitos hablados por una sola voz. Posteriormente, en el año 62, IBM presentó su Shoebox, con capacidad de entender 16 palabras habladas en Inglés.

Los años 70 significaron un punto de inflexión en el progreso de las tecnologías de reconocimiento de voz, al llamar la atención del Departamento de Defensa de los Estados Unidos, gracias a cuyos fondos fue posible el programa DARPA.

Figura 2.8 Dispositivo Shoebox de Reconocimiento de Voz, creado por IBM

De este programa surgió "Harpy", un sistema de reconocimiento capaz de entender 1011 palabras utilizando un método conocido como "beam search", una aproximación más eficiente al limitar la red de posibles frases. Además, durante la misma década los laboratorios Bell introdujeron otra mejora al posibilitar el reconocimiento de voz de múltiples individuos.

Los años 80 decantaron los sistemas de reconocimiento de voz hacia modelos de predicción al introducir el método estadístico conocido como modelo oculto de Markov. Al considerarse la probabilidad de que sonidos desconocidos fueran palabras, en lugar de usar plantillas de palabras y buscar patrones de sonido, se permitía la posibilidad teórica de reconocer un número ilimitado de palabras.

En 1985 surgió el programa de traslación de voz a texto Kurzveil, que daba soporte a un vocabulario de 5000 palabras, aunque trabajaba con dicción discreta, por lo cual era necesario pausar después tras cada palabra.

La compañía Dragon en los años 90, con la llegada de procesadores más poderosos al alcance del consumidor, crea softwares de reconocimiento como Dragon NaturallySpeaking lanzado a emcado en 1997. NaturallySpeaking reconocía discurso continuado de hasta 100 palabras por minuto, aunque tenía la desventaja de necesitar un proceso de entrenamiento del programa de 45 minutos y de tener un precio elevado de 695$.

En estos años el uso de reconocimiento de voz también saltó a aplicaciones en compañías telefónicas al crearse VAL de BellSouth como servicio de atención al cliente.

A partir de estos logros el progreso del reconocimiento de voz quedó estancado. Se alcanzó una precisión del 80% en el reconocimiento. El próximo gran salto lo dio Google con su aplicación para IPhone Google Voice Search [18].

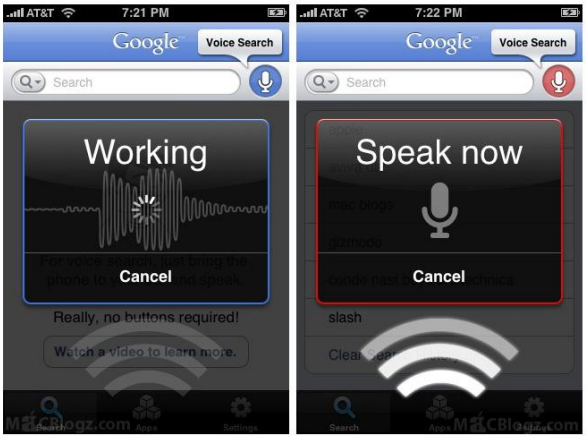

Figura 2.9 Aplicación para IPhone Google Search

La gran idea de Google fue que logró evitar el principal cuello de botella del reconocimiento de voz, la disponibilidad de datos, mediante el uso de un programa basado en nube. El procesado se realiza en sus centros de datos, utilizando todo su poder computacional y así liberando a los dispositivos móviles de toda carga de procesado. Esto además dio a Google su gran ventaja, la posibilidad de obtener grandes números de muestras de voz sin gran esfuerzo. Así pues, el sistema de búsqueda de Google por voz, en su versión Inglesa, incorpora a día de hoy más de 230 mil millones de palabras extraídas de consultas de usuarios.

En la actualidad los principales servicios abiertos al público de reconocimiento de voz están basados en este principio y profundamente vinculados a los dispositivos móviles. Los grandes avances previstos vienen en forma de cancelación de ruido y más mejoras de calidad.

2.4.2 Principales algoritmos

Los principales algoritmos de identificación actuales están basados en modelos estadísticos, de los cuales el modelado acústico y el modelado de lenguaje son una parte bastante importante [19].

El primero de estos algoritmos es el **modelo oculto de Markov**. Este modelo es generalmente utilizado para fenómenos que son estáticos y devuelve una secuencia de símbolos o cantidades. Dado que a cortos intervalos de tiempo el discurso humano puede ser tratado como estático, se puede interpretar como un modelo de Markov para muchas aplicaciones estocásticas.

Entrenar los procesadores basados en este modelo es sencillo y automático y los algoritmos son sencillos a nivel computacional.

Las **redes neuronales** fueron una opción atractiva para el modelado acústico hacia el final de los años 80. Desde entonces, este tipo de redes han sido utilizadas en aplicaciones de clasificación de fonemas, que son cada una de las unidades fonológicas mínimas que en el sistema de una lengua pueden oponerse a otras en contraste significativo, y reconocimiento de palabras aisladas.

A diferencia de los modelos de Markov, las redes neuronales no hacen presuposiciones sobre propiedades estadísticas y además permiten entrenamiento para discriminación de una manera eficiente y natural cuando se trata de estimar las probabilidades de un segmento de habla. Sin

embargo, las redes neuronales sufren en tareas de reconocimiento prolongadas y es por ello que en ocasiones se utilizan como preprocesado y apoyo de modelos ocultos de Markov.

El **Dynamic Time Warping** o DTW es la aproximación más tradicional al reconocimiento de voz, desplazada en la actualidad por el uso del modelo oculto de Markov. DTW es un algoritmo que mide la similitud entre dos secuencias que pueden variar tanto en tiempo como en velocidad. De esta manera el algoritmo es capaz de compensar con las diferentes velocidades de habla y permite a un procesador encontrar una coincidencia óptima entre dos secuencias dadas, con algunas restricciones.

2.4.3 Aplicaciones actuales

Las aplicaciones de reconocimiento más conocidas en la actualidad se basan en tecnologías móviles y de nube y están vinculadas a los principales sistemas operativos móviles. Así pues, Apple y su IOS disponen de Siri, Google y su SO Android de Google Search y Windows con su Windows Phone de Cortana.

Figura 2.10 Siri de Apple, Cortana de Windows y Google Search de Google

Estas aplicaciones requieren conexión a internet, siendo ésta la única desventaja que pueden presentar, dado que aprovechando la superior capacidad de procesado de los servidores externos y sus extensas bases de datos, el reconocimiento de voz solo mejora día a día.

Existen otros muchos módulos software no basados en nube, los cuales permiten una aproximación más tradicional al reconocimiento de voz a cambio de necesitar una mayor cantidad de procesado.

HTK o Kaldi ofrecen plataformas Open Source multiplataforma con algoritmos de Markov para el reconocimiento de voz que pueden ser integradas en diversos programas en forma de librerías C y C++ [20].

Módulos de software privado como Dragon Dictate o diversos módulos software incluidos como parte del sistema operativo Windows 7 se encuentran disponibles para usuarios de ordenadores actuales.

Figure 2-1 Dragon Dictate

Sin embargo en aplicaciones que requieren de portabilidad, los sistemas móviles siguen siendo dueños indiscutibles en cuanto a variedad de comandos reconocibles y capacidad de comunicación y vocabulario.

A pesar de ello, la creciente capacidad de los procesadores permite la creación de módulos como el EasyVR que, integrando diversos algoritmos de procesado en una pequeña placa de hardware dedicado, permite el reconocimiento de series de comandos en base a algoritmos más sencillos.

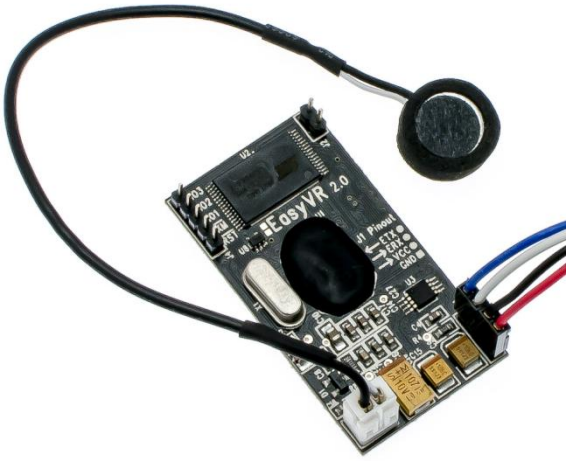

Figura 2.11 Módulo de reconocimiento de voz EasyVR

Este sencillo módulo de reconocimiento de voz resulta fácilmente integrable en aplicaciones con comandos limitados y es altamente compatible con Arduino, lo cual, junto con un precio reducido, 39€, y una gran facilidad para el entrenamiento de comandos, es una opción adecuada para ampliar las funcionalidades de PRESSMATIC

2.5 Herramientas de Diseño Vectorial

La interacción con el usuario es una parte esencial del control de cualquier dispositivo automatizado. A fin de facilitar la creación de aplicaciones, los kits de desarrollo tanto la pantalla integrada 4D como de Android disponen de herramientas y objetos prefabricados para realizar estos botones.

En el caso de PRESSMATIC, en el cual resulta indispensable que los interfaces sigan con mayor precisión las normativas de accesibilidad, las herramientas incorporadas en estos kits de desarrollo resultan insuficientes para dar los resultados deseados a la vez que se dotaba a las aplicaciones una imagen cohesionada. Es por ello por lo que se decidió utilizar una herramienta externa de diseño gráfico a fin de desarrollar los iconos de manera más acorde a la normativa.

Se sopesó el uso de diversas herramientas de diseño gráfico como los productos Adobe Photoshop y Adobe Illustrator. Illustrator parecía una gran opción dado que el dibujo vectorial es una metodologia facil y rápida de creación de Iconos y Logos muy utilizada en la actualidad. Al final, haciendo gala una funcionalidad similar a la ofrecida por el programa de diseño vectorial Adobe Illustrator, se decidió utilizar Inkscape.

Inkscape es un programa Open Source que tiene muchas funciones como maquetación web, diseño de logotipos, ilustraciones, etc. El diseño vectorial ofrece una gran ventaja con respecto al diseño tradicional dado que no pierde resolución al ampliar las imágenes. Sus formatos no basan las imágenes que produce en mapas de bits sino en vectores para realizar los dibujos. Aunque en el caso de los iconos PRESSMATIC esta gran cualidad no tiene tanto uso, la variedad de herramientas que ofrece el programa decantó la balanza a su favor.

Finalmente, Inkscape tiene una curva de aprendizaje no muy inclinada y además ofrece una gran cantidad de tutoriales en su web que permiten trabajar a un buen nivel rápidamente y en la plataforma de elección del diseñador gracias a su capacidad multiplataforma [21].

Es por ello que se escogió Inkscape como programa para el diseño de los iconos PRESSMATIC

Capítulo 3

3. Integración y Protocolo Bluetooth para PRESSMATIC

Para entender el funcionamiento de la conexión con el dispositivo PRESSMATIC en todos sus aspectos y permitir la expansión de funcionalidades a partir de éste proyecto, el presente capítulo tiene como finalidad presentar una visión general del proyecto junto con la arquitectura y los métodos utilizados para su desarrollo.

El desarrollo de un control Bluetooth paralelo a la creación de un dispositivo es un trabajo en constante cambio debida la necesidad última de adaptarse al funcionamiento básico del dispositivo PRESSMATIC y su pantalla.

3.1 Resumen de modos de operación de PRESSMATIC

A fin de entender adecuadamente como se crea el protocolo Bluetooth y la necesidad de los comandos creados, es necesario entender el funcionamiento básico del dispositivo PRESSMATIC

El dispositivo PRESSMATIC es un dispositivo electromecánico automático cuyo objetivo es asistir al movimiento de pinzado. Es por ello que PRESSMATIC ofrece una notable variedad de modos de operación para ajustarse a las necesidades y movimientos que requieran de la realización del pinzado.

El **modo de operación continuo** consiste en un movimiento de apertura y cierre constante sobre el que se tiene control sobre la velocidad. Este movimiento resulta tremendamente útil para objetos que requieran una acción continuada, como, en el caso aplicado a PRESSMATIC, unas tijeras. Así pues el usuario, además, controla el momento de inicio y parada del movimiento.

El **modo de operación paso a paso** consiste en un control manual del avance o retroceso del movimiento de pinzado a base de impulsos correspondientes a un paso del motor paso a paso integrado en el dispositivo. Este modo resulta muy adecuado para tareas de precisión y de agarre, por lo cual se integra directamente en el modo pinzas, tanto para las pinzas grandes como para las pequeñas.

El **modo de operación de corte único** consiste en un único movimiento de pinzado temporizado. Al activar el modo el usuario, PRESSMATIC cierra su cabezal por completo hasta topar con la interrupción analógica y lo abre ligeramente para parar tras un pequeño periodo de tiempo. Este modo resulta adecuado para tareas como el corte de uñas, en el cual se requiere únicamente un movimiento de pinzado antes de volver a ajustar la posición del aparato.

Por último el **modo de operación ciclos** consiste en un modo al cual se especifica el número de ciclos de apertura y cierre que se desea que el dispositivo haga junto con una confirmación para la activación. Este modo fue retirado de la implementación de interfaz inicial, pero sigue disponible como parte de la programación PRESSMATIC.

3.2 Comandos Bluetooth para PRESSMATIC

Dado que la comunicación con el módulo Bluetooth escogido para este proyecto utiliza una emulación de puerto serie RFCOMM, es importante explicar el funcionamiento de un puerto serie estándar.

La comunicación serial tradicional es un interfaz físico, típicamente vinculado al estándar RS-232, en el cual se transmite un bit de información a la vez en oposición a los puertos paralelos, que envían varios bits de datos simultáneamente. Para enviar los datos a través de esta conexión se utiliza un formato por el cual se crean paquetes de datos y se envían a una velocidad determinada a través de esta conexión.

La velocidad del puerto y del dispositivo han de coincidir siempre para que la conexión transmita con éxito. Esta velocidad se mide en baudios, que representa el número de símbolos enviados por segundo a través de la conexión. En este caso los símbolos son señales binarias, por lo cual el ratio de baudios por segundo equivale al ratio de bits por segundo.

Así pues, los datos se envían en grupos de un byte junto con dos o tres bits de enmarcado, que son uno o dos bits de parada y el bit de paridad. El bit o bits de parada indican cuando se ha finalizado el envío del paquete o caracter y corresponde al último elemento enviado del paquete de datos. Su función es permitir la resincronización del flujo de caracteres. El bit de paridad es utilizado como parte del método de paridad para detectar errores en la transmisión. El objetivo de este bit es hacer que el número de bits 1 contenido en el caracter, incluido el bit de paridad, sea par. Si no fuese así el dato estaría corrompido, aunque sí es cierto que un número par de errores pasan desapercibidos a éste método. Aunque existe la posibilidad de uso de éste método, generalmente se utiliza una configuración del paquete de datos 8-N-1, que significa, 8 bits, sin bit de paridad y con un bit de parada, dejando la corrección de errores al protocolo de comunicación. Para preparar al dispositivo al otro lado del puerto para recibir datos, los paquetes de datos son precedidos siempre por un bit de comienzo que indica el inicio de una transmisión de datos.

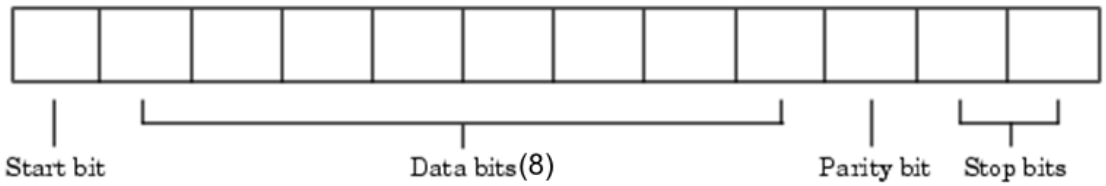

Figura 3.1 Estructura de un paquete de datos transmitido por puerto serie

Así pues, inicialmente se planteó la necesidad de mandar datos como la velocidad del Motor Paso a paso, el modo de operación y otros comandos en forma de enteros permitiendo un control interno a un nivel muy bajo y con un alto grado de precisión. Esto representaba un proceso de transmisión más pesado que requería el envío de varios paquetes y además de código en el software de PRESSMATIC que extrajera los valores del envío, lo cual era poco eficiente y podía llegar a ralentizar las acciones del dispositivo.

```
void loop(){
  if(dataCollected){
    Serial.flush();
    Serial.print("Data Recieved");
    dataTransform2();//Sends thedata recieved to its adequate variables
    dataPrint(); //Prints the data for debugging puropuses
    dataCollected = false;
    for(int i = 0; i<20; i++){
      data[i] = 0;//Data stored deleted
  }
}
//Arduino interrupt based on the arrival of data to the serial port
void serialEvent() {
  while(Serial.available()){
    //Obtain data from the serial
    char serialChar = (char)Serial.read();
    byte serialByte = (byte)Serial.read();
    Serial.print(serialByte);
    //Translate data to array
    data[count] = serialChar;
    dataByte [count] = serialByte;
    count++;
    if (serialChar == '\n'){
      dataCollected = true;
      count=0;
    }//End of the serial data
    Serial.flush();
  }
}
...
//This function expects the arrival of 3 bytes from the serial.
void dataTransform2(){
  //Translates the values from dataByte to its proper global variables
  Caso= int(dataByte[1]);
  Velocidad= int(dataByte[2]);
  Ciclos = int(dataByte[3]);
}
```

Este pequeño código de prueba ejemplifica uno de los primeros intentos de crear un envío de datos más detallado por comunicación serial. En él, al recibir datos seriales, se activa la interrupción SerialEvent que recoge los datos recibidos a través del serial en 2 arrays de datos. Estos datos posteriormente se trasladan a las variables destinadas a almacenar los distintos datos. El código no está ni mucho menos optimizado, aunque dado el número de ciclos necesarios, el uso de interrupciones como SerialEvent() y el uso de enteros aun optimizando el código iba a ralentizar el dispositivo fuera de los márgenes aceptables.

Añadido a lo previamente expuesto, se apreció, a lo largo del desarrollo del proyecto, que un usuario final no requería de tales niveles de precisión y que era preferible conformar un conjunto de comandos más cerrado e intuitivo. Es por ello, que se pasó a mandar una serie limitada de comandos integrados en caracteres, lo cual además requeriría del envío de un único

paquete de datos a través del puerto serie. Estos caracteres serían enviados al dispositivo a través del protocolo RFCOMM y recibidos en el puerto serie 3 de la placa integrada del dispositivo. En el dispositivo, un mucho más eficiente "switch-case" realiza las acciones adecuadas en función del comando recibido.

Éste enfoque intuitivo afectó únicamente al desarrollo de los interfaces, por lo que aquellos modos que compartan proceso de funcionamiento compartirán también comandos para el Bluetooth.

Grupo	Control	Comando
Menú de selección	Modo Continuo	b
	Modo Paso a Paso	c
	Modo Cortauñas	e
	Modo Ciclos	d
	Bluetooth	f
Modo Continuo	Lento	5
	Medio	6
	Rápido	7
	Start/Stop	0
Modo Cortauñas	Cortar Una	8
Modo Paso a Paso	Abrir	1
	Cerrar	2
Bluetooth	Desconectar	9
Común a todos los modos	Atrás/Inicio	a
Modo Ciclos(Actualmente no incluido en la interfaz)	Sumar un Ciclo	3
	Iniciar proceso de apertura y cierre	4

Tabla 3.1 Comandos Bluetooth para PRESSMATIC

A fin de ofrecer además un seguimiento de los comandos recibidos en el exterior desde la pantalla se añade el siguiente código dentro del programa de control PRESSMATIC, destinado a recibir los caracteres correspondientes a los comandos a través del Bluetooth y ejecutar las acciones necesarias en PRESSMATIC.

```
//If Bluetooth Serial data Available
   if (Serial3.available() > 0){
     dato_bluetooth = Serial3.read();//Read Data from Serial
     Serial3.println(dato_bluetooth);
     switch (dato_bluetooth){
        case 'a': form=6;  //Main menu
                  genieWriteObject(GENIE_OBJ_FORM, 0x00, 0x00);
                  start=0;
                  inicio=0;
                  break;
```

```
        case 'b': form=1;  //Continuous Operation Mode
                  genieWriteObject(GENIE_OBJ_FORM, 0x01, 0x01);
                  inicio=0;
                  break;
        case 'c': form=2;  //Stepper Operation Mode
                  genieWriteObject(GENIE_OBJ_FORM, 0x02, 0x02);
                  start=0;
                  inicio=0;
                  break;
        case 'd': form=3;  //Cicles Operation Mode
                  genieWriteObject(GENIE_OBJ_FORM, 0x03, 0x03);
                  break;
        case 'e': form=4;  //Nail Clipper Operation Mode
                  genieWriteObject(GENIE_OBJ_FORM, 0x04, 0x04);
                  break;
        case 'f': form=5;  //Bluetooth Menu
                  genieWriteObject(GENIE_OBJ_FORM, 0x05, 0x05);
                  break;
        case 'g': form=0;  //Internal Reset
                  genieWriteObject(GENIE_OBJ_FORM, 0x00, 0x00);
                  break;
        //Buttons
        case '0': led_est = !led_est;        // Start/Stop buttton
                  if(led_est==HIGH){
                    start=1;
                    genieWriteObject(GENIE_OBJ_LED, 0x00, 0x01);     // Write to
LED Digits 1
                    genieWriteStr(0,FRASE1);
                    }
                  if(led_est==LOW){
                    start=0;
                    genieWriteObject(GENIE_OBJ_LED, 0x00, 0x00);     // Write to
LED Digits 0
                    genieWriteStr(0,FRASE2);
                    }
                  break;
        case '1': boton1='1';                //Stepper Mode open
                  break;

        case '2': boton2='1';                //Stepper mode close
                  break;
        case '3': N_ciclos = N_ciclos + 1;   //Sums 1 cicle in Cicles mode
                 genieWriteObject(GENIE_OBJ_LED_DIGITS, 0x00, N_ciclos);
                  break;
        case '4': inicio=!inicio;            //Start the cicles operation
                  break;
        case '5': SetPinFrequencySafe(step_motor, Vlenta); //Slow speed
                  digitalWrite(MS1,HIGH);
                  break;
        case '6': SetPinFrequencySafe(step_motor, Vnormal);//Medium Speed
                  digitalWrite(MS1,LOW);
                  break;
        case '7': SetPinFrequencySafe(step_motor, Vrapida);//High Speed
                  digitalWrite(MS1,LOW);
                  break;
```

```
        case '8': cortar_una = '1';                          //Cut once in nail
clipper mode
                  break;
        default: form=0;
                 break;
     }
}
```

Este código representa la comprobación y acciones realizadas al recibir un comando Bluetooth, tal como se especifica en el siguiente flujograma.

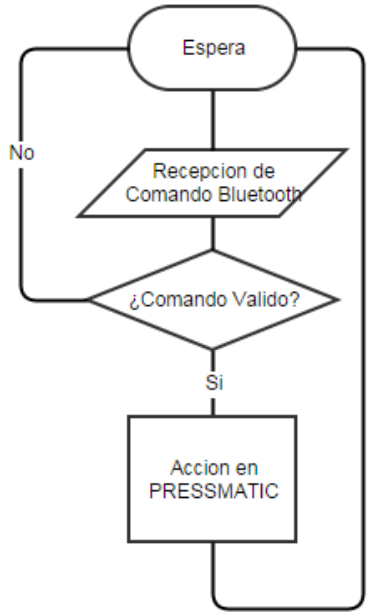

Figura 3.2 Flujograma de Recepcion de comandos Bluetooth en PRESSMATIC

`Serial3.available()` devuelve el número de bytes disponibles en el puerto serial, así pues, cuando es mayor que 0, hay datos disponibles. Estos datos se almacenan en `dato_bluetooth`, que posteriormente se pasa al "switch-case" hasta que ya no queden más datos en el puerto serial. La función `genieWriteObject()` es la encargada de modificar el estado de la pantalla integrada de tal manera que siga los comandos que llegan a través del Bluetooth. `start` es una variable encargada de habilitar o inhabilitar el motor dependiendo del modo. `inicio` forma parte del modo ciclos, que aunque ya no forma parte de la interfaz de usuario, sigue formando parte de la funcionalidad PRESSMATIC, disponible para futuros usos.

3.2 Configuración del módulo Bluetooth HC-05 a través de Arduino

El módulo HC-05 se puede configurar a diversos niveles y para adaptarse a las necesidades del mediante los comandos AT incluidos en la hoja de datos del módulo. A continuación se explica el proceso de configuración y los programas utilizados.

Para iniciar el proceso de configuración es necesario poner a nivel alto el pin de entrada WAKEUP. Al inicializarse el módulo con estas condiciones, la comunicación por puerto serie del módulo se pone por definición a 38400 baudios, por lo que es necesario, para poder configurar el módulo, iniciar la comunicación serie de Arduino con esa velocidad.

El programa básico utilizado para la configuración viene crea un sencillo puente entre el ordenador y el módulo HC-05, permitiendo usar el monitor serial del entorno de desarrollo Arduino para enviar los comandos AT.

El primer comando a mandar es `AT.` Este comando confirma si de hecho el módulo se halla preparado para recibir comandos. Una respuesta de `OK` indica que se puede continuar con la configuración.

Los comandos que inmediatamente resultan más interesantes son `AT+ROLE=<Param>`, `AT+NAME=<Param>` y `AT+PSWD=<Param>`. Estos tres comandos permiten respectivamente establecer el rol del dispositivo como maestro o esclavo, el nombre del dispositivo y la contraseña del dispositivo.

El módulo integrado en el dispositivo PRESSMATIC debe de ser esclavo, permitiendo así que los diversos dispositivos de control posible como Smartphones que ejerzan de maestro para el dispositivo y liberarán de éste la carga de realizar los distintos procedimientos de conexión. Para ello se manda a través de la consola el comando `AT+ROLE = 0`, aunque este forma parte de la configuración estándar.

Además, resulta interesante por seguridad cambiar la contraseña a un número diferente de los estándar 0000 o 1234 y establecer como nombre PRESSMATIC.

A fin también de usar la configuración óptima de baudios para trabajar con el Arduino, que es de 9600 baudios, es necesario utilizar el comando `AT+UART=<Param1>,<Param2>, <Param3>`. El parámetro uno corresponde al valor decimal de la señal en baudios, el dos al número de bits de parada y el tres al bit de paridad.

Así pues, se trabajará con Arduino utilizando la configuración `AT+UART=9600,0,0` .

Este modo de trabajo con seguridad básica utilizando el paradigma de seguridad de Bluetooth en Modo 2 ofrece un cierto grado de seguridad a la vez que asegura que no haya incompatibilidades debido a las distintas opciones de encriptado que ofrece el dispositivo.

3.3 Comunicación Maestro-Esclavo entre dos módulos Bluetooth HC-05

Debido a las diversas cualidades del módulo HC-05 que lo hacen adecuado para su uso en el dispositivo PRESSMATIC, resultaba interesante usarlo también para manejar la conectividad de los posibles módulos externos. Como se explica más adelante, el módulo HC-05 forma parte del prototipo de control basado en EasyVR y en Arduino, por lo que es necesario aclarar de qué manera es necesario configurarlo para que realice la conexión con PRESSMATIC.

Siguiendo el paradigma de creación orientada al usuario, el objetivo del módulo ha de ser la conexión inmediata con PRESSMATIC, sin pasos intermedios y sin intervención del usuario.

El proceso requiere pues la configuración del módulo como maestro, añadir el módulo implementado en PRESSMATIC a la lista de dispositivos vinculados del módulo y formalizar una conexión automática al encontrarse ambos dispositivos activos y dentro del rango.

Para ello es necesario saber, primero, la dirección del dispositivo al que pretendemos vincular el maestro. Haciendo uso del código de comunicación con el módulo utilizamos el comando `AT+ADDR?` para obtener la dirección del módulo integrado en PRESSMATIC.

```
Enter AT commands:
OK
+ADDR:2014:3:250263
OK
```

Figura 3.3 Uso de Comando AT para obtener la dirección del módulo

Una vez se consigue la dirección del módulo PRESSMATIC, se conecta el maestro al Arduino para realizar su configuración. Tras chequear que el módulo se encuentra en modo AT, se procede a configurarlo como maestro enviando el comando `AT+ROLE=1`. Una vez está configurado como maestro, procedemos a vincular o "bindear" el dispositivo. Para ello, se utiliza la dirección conseguida anteriormente y el comando `AT+BIND=<Param>`.

```
AT+ROLE=1

Enter AT commands:
OK
+ADDR:2014:3:250263
OK
+ROLE:0
OK
OK
```

Figura 3.4 Uso de Comandos AT para establecer el Rol como Maestro

Este tipo de conexión cumplirá exactamente el objetivo deseado, al permitir que ambos dispositivos se conecten automáticamente.

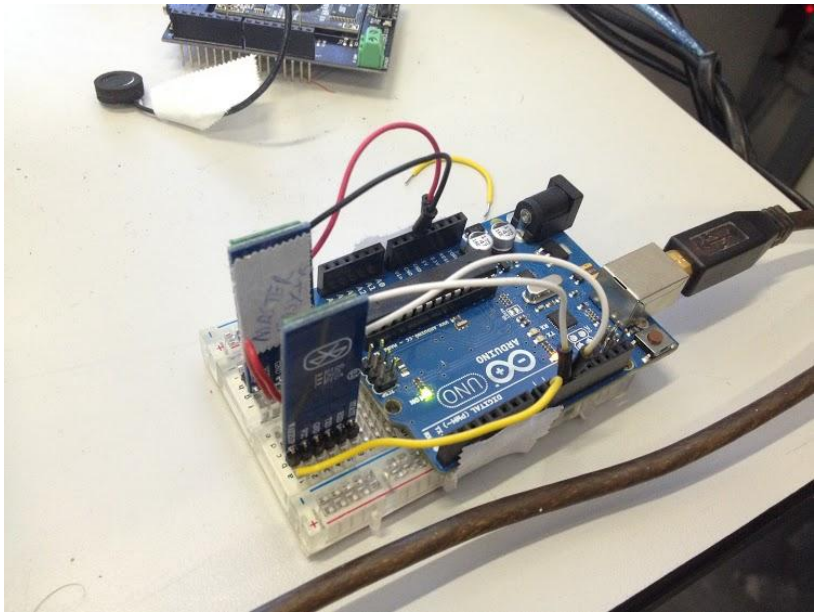

Figura 3.5 Prueba de comunicación entre dos módulos HC-05 conectados como Maestro-Esclavo

Por último, también es necesario adaptar siempre la velocidad de el puerto serie a la velocidad utilizada por el microprocesador base del módulo. En éste caso, dado que para el módulo externo se utiliza en conjunción con una placa Arduino, se utiliza el comando `AT+UART=9600,0,0` para adecuar la velocidad del puerto serie al Arduino.

Así pues, una vez asentado el protocolo y preparado el módulo Bluetooth de PRESSMATIC, es posible comenzar a elaborar las aplicaciones externas en base a este protocolo siguiendo los métodos de conexión aquí descritos.

Capítulo 4

4. Aplicaciones para la conectividad Bluetooth

En este capítulo se explican en detalle las aplicaciones basadas en el protocolo descrito en el capítulo 3, haciendo hincapié en cómo ha sido implementada cada conexión en su respectivo entorno de desarrollo.

4.1 Conectividad Bluetooth para la aplicación de Android PRESSMATIC

El entorno de desarrollo de Android presenta varias ventajas por las cuales fue elegido como entorno para la aplicación de PRESSMATIC. El mayor tamaño de la comunidad de programadores que utilizan Android, la capacidad multiplataforma del entorno de programación y la extensa documentación disponible hacen de él una elección ideal para desarrollar la aplicación PRESSMATIC, lo que, junto con la gran variedad de dispositivos que hacen uso de este sistema operativo, inclinó la balanza hacia este entorno. Sin embargo, la comunicación serial con el Android no resulta tan sencilla como la que se puede tener con Arduino.

Para crear una conexión Bluetooth estable en la aplicación y poder comunicarse con los dispositivos externos es necesario seguir un proceso estricto de creación de diversos objetos de comunicación.

Estos objetos se basan a su vez en cuatro objetos básicos para la comunicación Android: El `Thread`, el `Handler`, el `BluetoothSocket` y el `Bundle`.

Un `thread` es una unidad concurrente de ejecución o, aplicando una traducción literal, un "hilo" de ejecución. Al crear un `thread` independiente del `main thread` de ejecución de la aplicación, se permite el operar a la conexión bluetooth sin esperar al resto del ciclo de ejecución de la aplicación.

Un `handler` es un objeto que permite el envío y procesamiento de un mensaje del sistema y de objetos ejecutables asociado con la cola de mensajes de un `thread`. Cada `handler` va asociado a un `thread` independiente, al cual se vincula en el momento de la creación.

Un `socket` es utilizado generalmente para la creación de un receptáculo para la conexión con un protocolo IP o Ethernet del cliente. De esta manera es posible intercambiar datos entre programas situados en distintos dispositivos. En el caso de la aplicación PRESSMATIC©, el socket utilizado es un `BluetoothSocket`, que se crea para formalizar la conexión entre el módulo Bluetooth del propio dispositivo utilizando la dirección obtenida de la lista de dispositivos encontrados para conectarse a ella. De esta manera el socket se convierte en el medio por el cual la aplicación PRESSMATIC se comunica con el propio programa del dispositivo [22].

Por último se utiliza un `bundle` para englobar los paquetes de datos enviados a través de la conexión Bluetooth al utilizar la función `send()`.

En torno a estos cuatro objetos el objetivo de la aplicación es crear una conexión estable, que identifique si hay fallos de conexión o desconexiones del dispositivo remoto mediante una máquina de estados y ofrezca esta información al usuario.

4.1.1 Preparación de la Conexión

Para realizar la conexión con éxito, lo primero que es necesario confirmar, y que la aplicación asegura, es que el dispositivo en el cual se haya instalada la aplicación dispone de un módulo Bluetooth propio para realizar la conexión. Es importante destacar que, debido a la gran cantidad de diferentes dispositivos que utilizan Android, a pesar de que la gran mayoría de estos correspondan a smartphones que integran conectividad Bluetooth, no es posible asegurar con absoluta certeza que el dispositivo disponga de Bluetooth. Esto se asegura preguntando al sistema operativo Android si hay un adaptador Bluetooth disponible mediante el siguiente código:

```
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if (mBluetoothAdapter == null) {
            finishDialogNoBluetooth();
            return;
        }
```

El objeto `BluetoothAdapter` corresponde al módulo Bluetooth que utilizará el dispositivo para realizar la conexión. Mediante la función `getDefaultAdapter`, obtenemos un `handler` al módulo Bluetooth local. Así pues, si este objeto estuviera vacío, no habrá adaptador Bluetooth y por lo tanto se retornará un mensaje al usuario indicando la conexión como imposible.

En caso de que si existiera un módulo, se pedirá al usuario que active el Bluetooth antes de proceder a realizar cualquier acción dentro de la aplicación. Una vez activado el Bluetooth, al pulsar el icono con el símbolo de Bluetooth se inicia el proceso de conexión.

Figura 4.1 Proceso de activación de Bluetooth desde PRESSMATIC

Para crear una conexión, es necesario obtener la lista de dispositivos emparejados junto con la lista de nuevos dispositivos. Para ello se crea una actividad `ListaDispositivos` a la cual se llama desde la actividad principal, que corresponde a la primera pantalla.

Figura 4.2 Pantalla inicial de PRESSMATIC correspondiente a la actividad Pressmatic

El objeto `ListaDispositivos` se llama mediante el método onActivityResult, cuyo objetivo es el de iniciar una actividad a fin de recibir un resultado y una serie de datos.

```
Intent serverIntent = new Intent(Pressmatic.this, ListaDispositivos.class);
startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
```

Como se puede observar el método `startActivityForResult()` realizará el intent a la clase `ListaDispositivos`. Esto crea un display con la lista de dispositivos ya vinculados al terminal y además permite la búsqueda de nuevos dispositivos. Tras seleccionar uno de los dispositivos encontrados se manda la dirección en forma de string y además devuelve datos en forma de un `requestCode` y un `resultCode`.

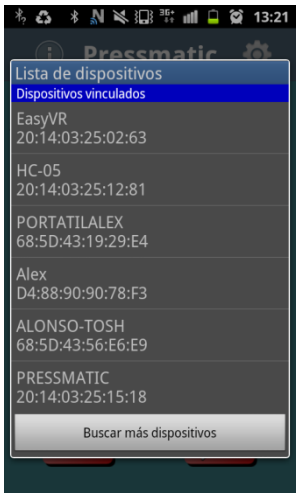

Figura 4.3 Actividad ListaDispositivos mostrando una lista de dispositivos vinculados

Con estos datos se crea un objeto `BluetoothDevice`, objeto contenido en la clase principal del servicio Bluetooth. Este objeto obtiene el dato de la dirección del módulo objetivo de la conexión al crearse para después ser transferido al método `connect()`.

Esto también devolverá el valor entero contenido en la variable `requestCode` REQUEST_CONNECT_DEVICE, junto con un valor de resultado, que será recibido por el método `onActivityResult()`. Este método, utilizando un "switch-case" en función del código de solicitud requestCode, iniciará el proceso de conexión al dispositivo deseado.

```
    /**
     * Con esta función se determina lo que se hace con la información
     * proveniente de elegir o no elegir un dispositivo, con el
     * resultado de dicha acción.
     * @param requestCode es el código que identifica el caso
     * @param resultCode puede ser afirmativo o positivo
     * @param data obtiene información extra
     */
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        switch (requestCode) {
            case REQUEST_CONNECT_DEVICE:
                // When DeviceListActivity returns with a device to connect
                if (resultCode == Activity.RESULT_OK) {
                    // Get the device MAC address
                    String address = data.getExtras()
                            .getString(ListaDispositivos.EXTRA_DEVICE_ADDRESS);
                    // Get the BLuetoothDevice object
                    BluetoothDevice device =
mBluetoothAdapter.getRemoteDevice(address);
                    MyApplication mApplication =
(MyApplication)getApplicationContext();
                    mApplication.setAddress(address);
                    mApplication.setBluetoothDevice(device);
                    // Attempt to connect to the device VIA SERVICE
                    startService(new
Intent(Pressmatic.this,BluetoothConexion.class));
                    Intent intent = new Intent(this, BluetoothConexion.class);
                    bindService(intent,mConnection, Context.BIND_AUTO_CREATE);
                    mFirstConnection = false;
                } else if (resultCode == Activity.RESULT_CANCELED){
                    EstadoBoton = false;
                    BluetoothButton.setChecked(EstadoBoton);
                }
                break;
            case REQUEST_ENABLE_BT:
                // When the request to enable Bluetooth returns
                if (resultCode != Activity.RESULT_OK) {
                    finishDialogNoBluetooth();
                }
        }
    }
```

Utilizando la función `getRemoteDevice(string String)`, se prepara a la aplicación para conectar creando el objetivo de la conexión o `BluetoothDevice`. Antes de conectar, tanto la dirección como el objeto `device` son enviados a `MyApplication` para pasar a ser variables globales.

Finalmente, una vez creados estos objetos, se realiza una llamada al método `startService(Intent intent)` que a su vez realiza una llamada al método `onStartCommand()` de la clase BluetoothConexion.

En él se toman las variables `Handler`, `Device` y `Address` de `MyApplication` y se utilizan para, mediante la función `connect` del dispositivo, realizar la conexión.

En caso de que este procedimiento de conexión falle se notificará al usuario mediante un mensaje Toast en la pantalla.

Asi pues, el conjunto del proceso de conexión puede ser representado mediante el siguiente Flujograma.

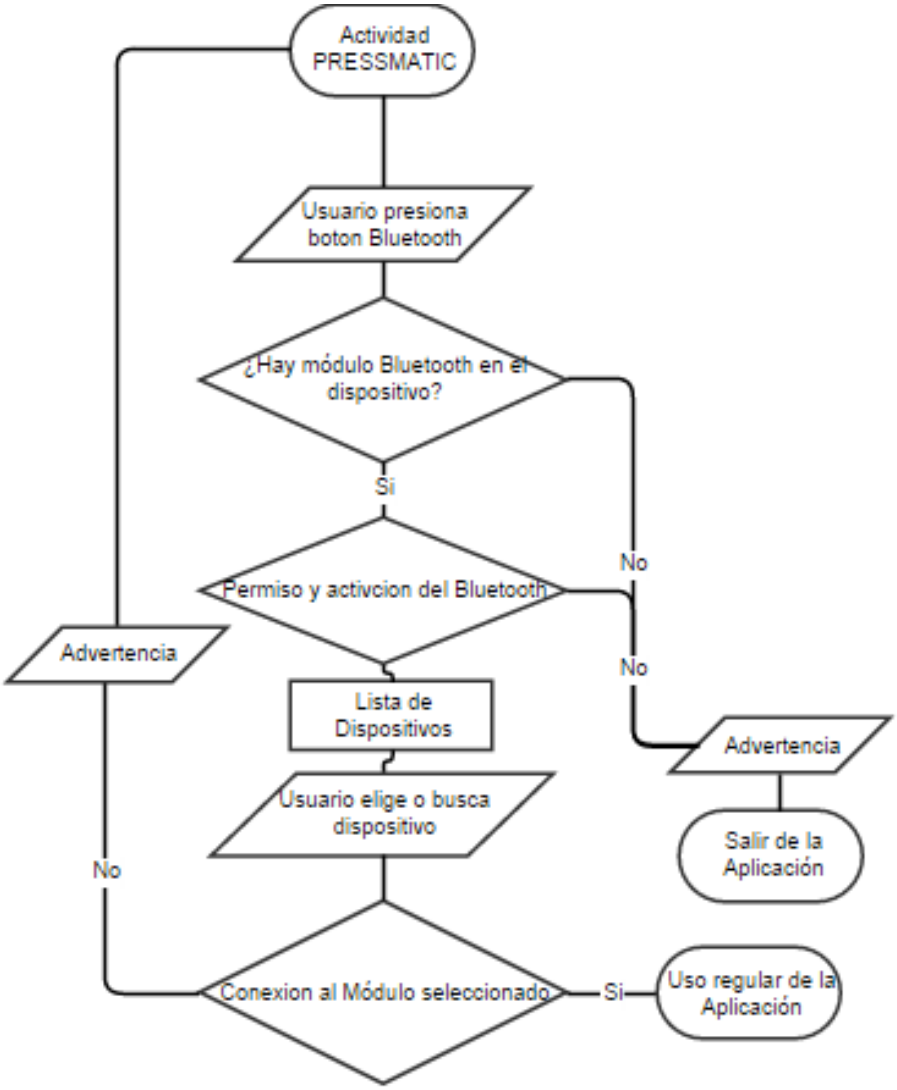

Figura 4.4 Diagrama de Flujo de conexión Bluetooth

Figura 4.5 Mensaje Toast confirmando la imposibilidad de realizar una conexión.

4.1.2 Funcionamiento interno del proceso de conexión

Es en la creación y mantenimiento de la conexión donde hubo mayores dificultades a la hora de escribir el código. El método de conexión utilizado en PRESSMATIC está basado en el código abierto de la aplicación BlueTerm [23]y en el ejemplo abierto de Android BluetoothChat, que crean la conexión Bluetooth como parte de la actividad principal. Como actividad se puede entender una ventana sobre la cual el programador crear un interfaz de usuario, permitiendo acciones e interacciones con la aplicación. Si se observa el ciclo de vida de las actividades en el sistema operativo Android, es posible apreciar porque esto supone un problema.

Debido a que el planteamiento de la aplicación PRESSMATIC requiere de la circulación a través de varias actividades, la creación de la conexión en una única actividad suponía que al cambiar de actividad la conexión se suspendiera o incluso destruyera al pasar por `onStop` u `onDestroy`. Por ello el código disponible no era adecuado, dado que el objeto que representa la conexión Bluetooth sólo existe dentro de la actividad principal. Al entrar esa actividad en pausa, la conexión Bluetooth se perdía, para no volver a recuperarse aunque se volviera a entrar en la actividad.

Esto dejaba una serie de opciones limitadas para mantener una conexión Bluetooth.

La primera idea que surgió fue simplemente restaurar la conexión en cada cambio de actividad mediante el uso del procedimiento de Blueterm para crear la conexión. Almacenando el dato de dirección elegida como una variable global a la aplicación se evitaba que el usuario volviera a necesitar elegir el dispositivo a través de la `DeviceList`. A pesar de que resultaba una opción funcional, no era ni por asomo óptima, sobre todo debido a la velocidad de restauración de la conexión, demasiado lenta como para ajustarse a los requerimientos del proyecto.

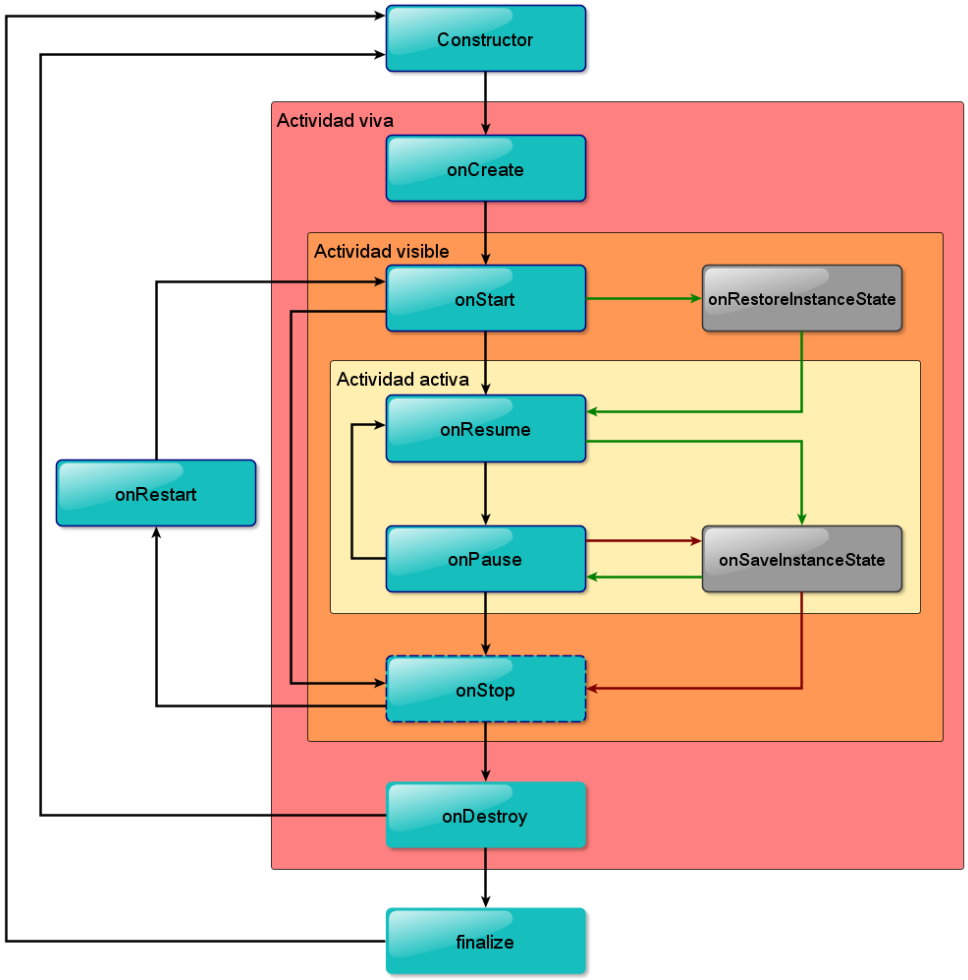

Figura 4.6 Ciclo de vida de una actividad en el sistema operativo Android

La necesidad de cambios de pantalla es parte inherente del propósito de la aplicación. Como ya se comenta en otras secciones de este escrito, las pantallas se han de mantener con un diseño sencillo y poco recargado de botones, permitiendo un control intuitivo y con escasos requerimientos de destreza manual, lo cual descartaba la idea de acumular todos los controles en una misma pantalla.

Así pues, quedaban dos opciones, la creación de un interfaz cambiante dentro de la propia actividad o la creación de la conexión dentro de un `service`.

La primera opción se ajustaba en parte a los propósitos del proyecto, pero fue descartada en beneficio de la segunda debido a las diversas dificultades técnicas que suponía.

El objetivo detrás de la primera opción era la creación de un interfaz parecido al de BlueTerm con su clase `terminalEmulator`. Esta clase crea un fragmento en la pantalla el cual se hace un display de los datos recibidos y enviados a través de la conexión Bluetooth, los cuales se representan imitando a una especie de consola o terminal MSDOS.

Esto planteaba problemas sobre cómo manejar los cambios de pantalla o como crear los distintos botones dentro de la interfaz cambiante. Se sopesó una máquina de estados interna para controlar estos cambios y en qué situación se hallaban activos, pero tras desarrollar la idea las

dificultades técnicas hicieron que el desarrollo del proyecto se inclinara hacia el uso de un `service`.

Un `service` dentro del entorno de Android es un componente de las aplicaciones que representa el deseo de realizar una operación que se ejecute en un plazo más largo que una actividad mientras no se interactúa con el usuario o para dar una determinada funcionalidad a la cual otras aplicaciones pueden acceder. Esto se ajusta al propósito de la aplicación, dado que se pretende, al crear la conexión Bluetooth como un `service`, aprovechar las propiedades de este objeto que sobre todo permite disgregar la conexión de las actividades de la aplicación y colocarla en un modo de ejecución independiente.

Es importante destacar como el ciclo de vida de un `service` difiere en gran medida del ciclo de vida de una actividad.

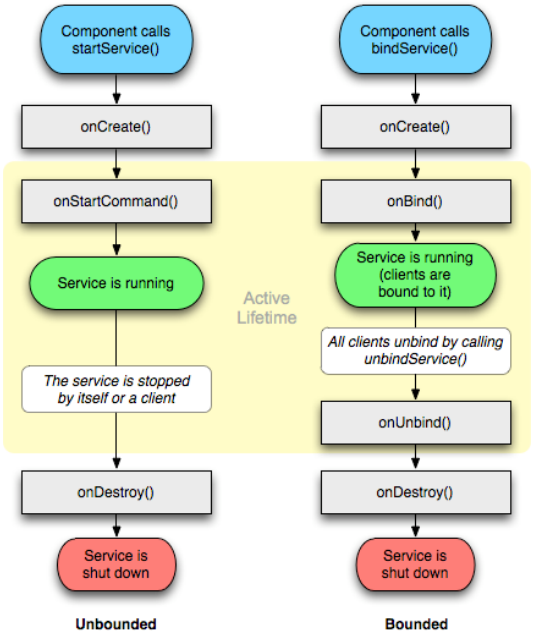

Figura 4.7 Ciclo de vida de un service en el sistema operativo Android

En función de cómo es inicializado el servicio, ya sea por el método `onStartCommand()` o por el método `onBind()`, el ciclo de vida es uno u otro. La gran diferencia radica sobre todo en la inicialización `onBind()`. Esta inicialización hace que la vida del service quede vinculada a la existencia de una actividad que esté activamente bindeada a ella. Esto significa que si todas las actividades que estuvieran conectadas al servicio de esta manera fueran destruidas, el servicio finalizará. En cambio, si se inicializa el service mediante `onStartCommand()`, la falta de vinculación de una actividad no cierra el service. Así pues se escoge este método de inicialización para asegurar que la conexión no se interrumpe excepto a petición del usuario.

El objeto `mBluetoothConexion` que contendrá una referencia al `service` se crea durante el `onCreate()` mediante la siguiente línea de código.

```
mBluetoothConexion = new BluetoothConexion(this, mHandlerBT);
```

Esto llama al constructor parametrizado siguiente:

```
public BluetoothConexion(Context context, Handler handler){
            mAdapter = BluetoothAdapter.getDefaultAdapter();
            mState = STATE_NONE;
            mHandler = handler;
            mContext = context;
            mAllowInsecureConnections = true;

        }
```

De esta manera se le envía al objeto recién creado el contexto de la aplicación y el handler que contendrá los distintos datos de comunicación con el dispositivo. Este `handler` ha de ser el mismo durante todo el uso de la aplicación, así pues recurriendo a la clase `MyApplication` ,que extiende a la clase `Android Application`, se pueden almacenar una serie de variables globales, como en este caso el `handler`.

Este objeto recién creado no representa la conexión ni inicializa el service, solo es un objeto destinado a contener la instancia del `service` dentro de la propia actividad.

La función `connect()` es la principal encargada de crear todos los objetos necesarios para formalizar la conexión y permitir la comunicación. Ésta función merece una explicación en profundidad, además de una mención necesaria a la máquina de estados utilizada para controlar el proceso.

Muchas de las funciones incluidas en `BluetoothConexion` modifican o responden all estado de una máquina de estados con 3 variaciones posibles. Estos son `STATE_NONE`, `STATE_CONNECTING` y `STATE_CONNECTED` y van asignados a un entero privado `mState`.

La función `connect` toma el objeto `BluetoothDevice` y, si el estado es `STATE_CONNECTING`, cancela todo `ConnectThread` y además en cualquier caso cancela todo `ConnectedThread` existente. Después de ello crea un nuevo `ConnectThread` y lo inicializa basándose en el device y cambia el estado a `STATE_CONNECTING` mediante la función setState. Es importante destacar que esta función es `syncronized`, de manera que sólo se podrá realizar una ejecución del método synchronized sobre el mismo objeto de manera simultánea, aunque se halle en diferentes threads. [24] Esto asegura la estabilidad de la conexión e impide errores de consistencia.

```
        /**
         * Start the ConnectThread to initiate a connection to a remote device.
         * @param device  The BluetoothDevice to connect
         */
        public synchronized void connect(BluetoothDevice device) {


            // Cancel any thread attempting to make a connection
            if (mState == STATE_CONNECTING) {
                if (mConnectThread != null) {mConnectThread.cancel();
mConnectThread = null;}
            }

            // Cancel any thread currently running a connection
```

```
            if (mConnectedThread != null) {mConnectedThread.cancel();
mConnectedThread = null;}

            // Start the thread to connect with the given device
            mConnectThread = new ConnectThread(device);
            mConnectThread.start();
            setState(STATE_CONNECTING);

        }
```

Al inicializar el `ConnectThread` se crean dos variables privadas internas y `final` que almacenan y retienen el `device` y el `socket`. De esta manera, una vez se asignan valores a estas variables, al ser `final`, no pueden ser modificados. El constructor, en función de si se permiten las conexiones inseguras o no, realiza una llamada a un método u otro para obtener el `BluetoothSocket` deseado y posteriormente almacenarlo en la variable final.

Tras ser construido, la función `start()` aplicada a este objeto llama al método `run()` de la clase `ConnectThread`. Este método realiza como primera medida una cancelación del proceso de búsqueda de dispositivos Bluetooth a fin de agilizar la conexión con el dispositivo elegido.

```
public void run() {
                setName("ConnectThread");
                // Always cancel discovery because it will slow down a connection
                mAdapter.cancelDiscovery();
                // Make a connection to the BluetoothSocket
                try {
                    // This is a blocking call and will only return on a
                    // successful connection or an exception
                    mmSocket.connect();
                } catch (IOException e) {
                    connectionFailed();
                    // Close the socket
                    try {
                        mmSocket.close();
                    } catch (IOException e2) {

                    }
                    // Start the service over to restart listening mode
                    //BluetoothSerialService.this.start();
                    return;
                }

                // Reset the ConnectThread because we're done
                synchronized (BluetoothConexion.this) {
                    mConnectThread = null;
                }
                // Start the connected thread
                connected(mmSocket, mmDevice);
            }
```

Posteriormente se hace una llamada a un método "try- catch" que incluye la conexión del Socket en su interior. Los métodos "try-catch" están pensados para obtener Excepciones de ejecución durante el propio runtime y actuar en consecuencia. En caso de que esta conexión falle, la excepción llamara a la función `connectionFailed()` y a un segundo método "try-catch" que cerrará el socket.

En caso de que la conexión tenga éxito se resetea el `ConnectThread` del objeto `BluetoothConexión`, que es una copia del `this` o el objeto sobre el que se trabaja, y se pasa a llamar a la función `connected`, a la cual se le pasan el `mmSocket` y el `mmDevice`.

De nuevo, la función `connected` es `syncronized`, y se cerciora de que no haya ningún `connectThread` ni `connectedThread` en activo. Posteriormente crea un thread `ConnectedThread` al cual se envía el socket y es inicializado.

```
         /**
         * Start the ConnectedThread to begin managing a Bluetooth connection
         * @param socket  The BluetoothSocket on which the connection was made
         * @param device  The BluetoothDevice that has been connected
         */
        public synchronized void connected(BluetoothSocket socket,
BluetoothDevice device) {
            // Cancel the thread that completed the connection
            if (mConnectThread != null) {
                mConnectThread.cancel();
                mConnectThread = null;
            }
            // Cancel any thread currently running a connection
            if (mConnectedThread != null) {
                mConnectedThread.cancel();
                mConnectedThread = null;
            }
            // Start the thread to manage the connection and perform
transmissions
            mConnectedThread = new ConnectedThread(socket);
            mConnectedThread.start();

            // Send the name of the connected device back to the UI Activity
            Message msg = mHandler.obtainMessage(Pressmatic.MESSAGE_DEVICE_NAME);
            Bundle bundle = new Bundle();
            bundle.putString(Pressmatic.DEVICE_NAME, device.getName());
            msg.setData(bundle);
            mHandler.sendMessage(msg);
            setState(STATE_CONNECTED);
        }
```

Además de inicializar el `thread`, se crea un nuevo `bundle` con el que se envían los datos del dispositivo al que se formaliza la conexión mediante el `mHandler` y la función `sendMessage()`.

El constructor `ConnectedThread`, de nuevo, crea un `socket`, pero en este caso además crea dos objetos, un `InputStream` y un `OutputStream`, que se crean en base al `socket` mediante

el método "try-catch" combinado con las funciones `getInputStream()` y `getOutputStream()` del socket que se pasa al constructor.

Una vez creado el `thread`, este se inicializa. En la inicialización se crea un buffer de bytes en el cual se almacenan las señales de entrada desde el socket y una variable bytes que almacena los datos entrantes en forma de entero. Esto es posible mediante el uso de la función read, método perteneciente a la clase `InputStream` y al cual se pasa como variable el buffer.

```
public void run() {

                byte[] buffer = new byte[1024];
                int bytes;

                // Keep listening to the InputStream while connected
                while (true) {
                    try {
                        // Read from the InputStream
                        bytes = mmInStream.read(buffer);


                        //mEmulatorView.write(buffer, bytes);
                        // Send the obtained bytes to the UI Activity
                        //mHandler.obtainMessage(BlueTerm.MESSAGE_READ, bytes, -
1, buffer).sendToTarget();
                    } catch (IOException e) {

                        connectionLost();
                        break;
                    }
                }
```

Este buffer también es variable para la función `write`, que es la encargada, usando el objeto `OutputStream`, de enviar datos a través de la conexión.

Una vez terminado este procedimiento, la conexión queda formalizada y es un servicio estable. Así pues, se pasa a "bindear" o vincular, entendiéndose por "bindear" asociar una `Activity` a un `Service` determinado, la actividad principal al servicio mediante la función `bindService()`.

4.1.3 Vinculando Actividades al Servicio

El proceso de vinculación de la actividad es mucho más corto que el de creación de la conexión, y depende solamente de el método `onBind()` de la clase `BluetoothConexion` y de su clase derivada `LocalBinder`. Dentro de `BluetoothConexion` se crea un objeto de la clase `IBinder` que es un nuevo `LocalBinder`. El objeto `mBinder` solo tiene un método asignado, que es el `getService()`, que retornará el objeto `BluetoothConexion` que representa el servicio activado. La función `onBind()` además de retornar este objeto `mBinder`, tomará el `Handler` general desde `MyApplication` y lo igualará al de la actividad bindeada, permitiendo así el flujo de datos entre la actividad y el `service`.

Para la ejecución del método `bindService` es necesario también la creación de un objeto `ServiceConnection` en cada una de las actividades que pretendan vincular al servicio. Este

objeto contiene dos métodos: `onServiceConnected()`, encargado de igualar la instancia del `service` en la actividad a la del `service` ya creado y de cambiar el booleano `mBound` a `TRUE` para así confirmar la conexión, y `onServiceDisconnected()`, que vuelve a poner `mBound` a `FALSE`.

```
    /**
    * Defines callbacks for service binding, passed to bindService()
    */
    private ServiceConnection mConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName className, IBinder service)
{
            BluetoothConexion.LocalBinder binder =
(BluetoothConexion.LocalBinder) service;
            mBluetoothConexion = binder.getService();
            mBound = true;
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {
            mBound = false;
        }
    };
```

Por último, es necesario pasarle al método el flag `BIND_AUTO_CREATE`, para que el servicio se cree automáticamente mientras exista la vinculación. A pesar de que en este caso esto no es estrictamente necesario, funciona como una medida de seguridad al crear de nuevo el servicio si este fuera destruido por algún casual.

Así pues, tras la vinculación, el objeto `mBluetoothConexion` de la actividad `Pressmatic` equivaldrá al del propio `service`, permitiendo así una interacción con el dispositivo conectado.

Figura 4.8 Conexión con éxito

En caso de que esa conexión se perdiera en algún punto de la ejecución del programa, aparecerá el siguiente mensaje.

Figura 4.9 Mensaje Toast de pérdida de conexión

4.1.4 El Servicio, su permanencia y el ciclo de vida de las Actividades

Lo descrito hasta ahora representa un ciclo de conexión inicial, incluyendo la selección del dispositivo, la creación del servicio y la vinculación a ese servicio. Pero es notable destacar que , debido a la naturaleza de la aplicación PRESSMATIC, la relación con el `service` se halla presente a lo largo de todo el ciclo de vida de las actividades que conforman la aplicación.

Un cambio de actividad representa la creación de una nueva actividad y la destrucción o pausa de una antigua. Por ello es necesario desvincular la actividad saliente y vincular la entrante al `service`.

Si se toma como ejemplo el paso de la actividad `Pressmatic` a la actividad `ModoContinuo`, primero se comienza llamando al método `onCreate()` de la actividad `ModoContinuo`, que crea todo la interfaz junto con los botones correspondientes la aplicación. Siguiendo el ciclo de vida de la aplicación, se pasa al `onStart()`, donde mediante el método `bindService()` se vincula la actividad al servicio tal y como se explicó anteriormente. Posteriormente se ejecuta el onPause y la nueva actividad pasa a estar en primer plano de la aplicación. Ahora que ha pasado a segundo plano, se llama al método `onStop()` de la actividad `Pressmatic`. Este método, si `mBound` es `TRUE`, llamará al método `unbindService(Connection mConnection)` que desvinculara la actividad del servicio sin mayor repercusión para éste último y la variable `mBound` pasará a ser falsa. Por último, en función de los requisitos de memoria del terminal, la actividad será destruida o no.

Siguiendo con el ejemplo, si, al hallarnos en la actividad `ModoContinuo` ,se desea volver a `Pressmatic`, el proceso es idéntico al anterior exceptuando un detalle respecto al `onStart()`. Debido a que, como parte de los objetivos de la aplicación , es necesario que la actividad bindee sola, sin ninguna intervención por parte del usuario, una vez se ha creado el servicio, fue necesario crear una variable bool que asegurara que el método `onStart()` no creara el `service` sin haber recibido los datos necesarios, puesto que esto produciría un crash. Dado que la propia aplicación prohíbe cambios de actividad si no hay un servicio creado, es posible asumir con seguridad que una vez se ha efectuado un primer `onStart()` ya hay un servicio creado en la segunda ocasión que se ejecute éste método.

```
@Override
    public void onStart() {
        super.onStart();
            if(!mFirstConnection){
            Intent intent = new Intent(this, BluetoothConexion.class);
            bindService(intent,mConnection, Context.BIND_AUTO_CREATE);
            }
        mEnablingBT = false;
    }
```

Así pues, la primera vez que se ejecuta el `onStart()` de `Pressmatic` la variable booleana `mFirstConnection` pasa a ser falsa y en las subsiguientes ejecuciones se producirá la vinculación en el `onStart()`.

Este servicio se mantiene a lo largo de toda la ejecución del programa, e incluso tras salir de él. A fin de salvaguardar la integridad y continuidad del servicio, se ha puesto como condición necesaria que este sea interrumpido manualmente a través del menú y que no esté vinculado a ningún ciclo de vida de ninguna actividad.

En caso de que haya una instancia del servicio conectada a un dispositivo, accediendo al menú desde `Pressmatic` aparecerá la opción de desconectar, que llamará al método `stop()` de `BluetoothConexion`. Éste método, además de eliminar todos los `threads` pertinentes, incluye entre sus llamadas la función `stopSelf`, que llamada dentro de una función perteneciente al servicio lo destruye.

Así pues, la aplicación quedará lista para cerrarse y acabar su ejecución.

A fin de ofrecer una visualización del papel de la clase BluetoothConexion en la aplicación PRESSMATIC, se presenta el siguiente diagrama de clases, que corresponde a la totalidad de las actividades y clases utilizadas y sus relaciones entre ellas.

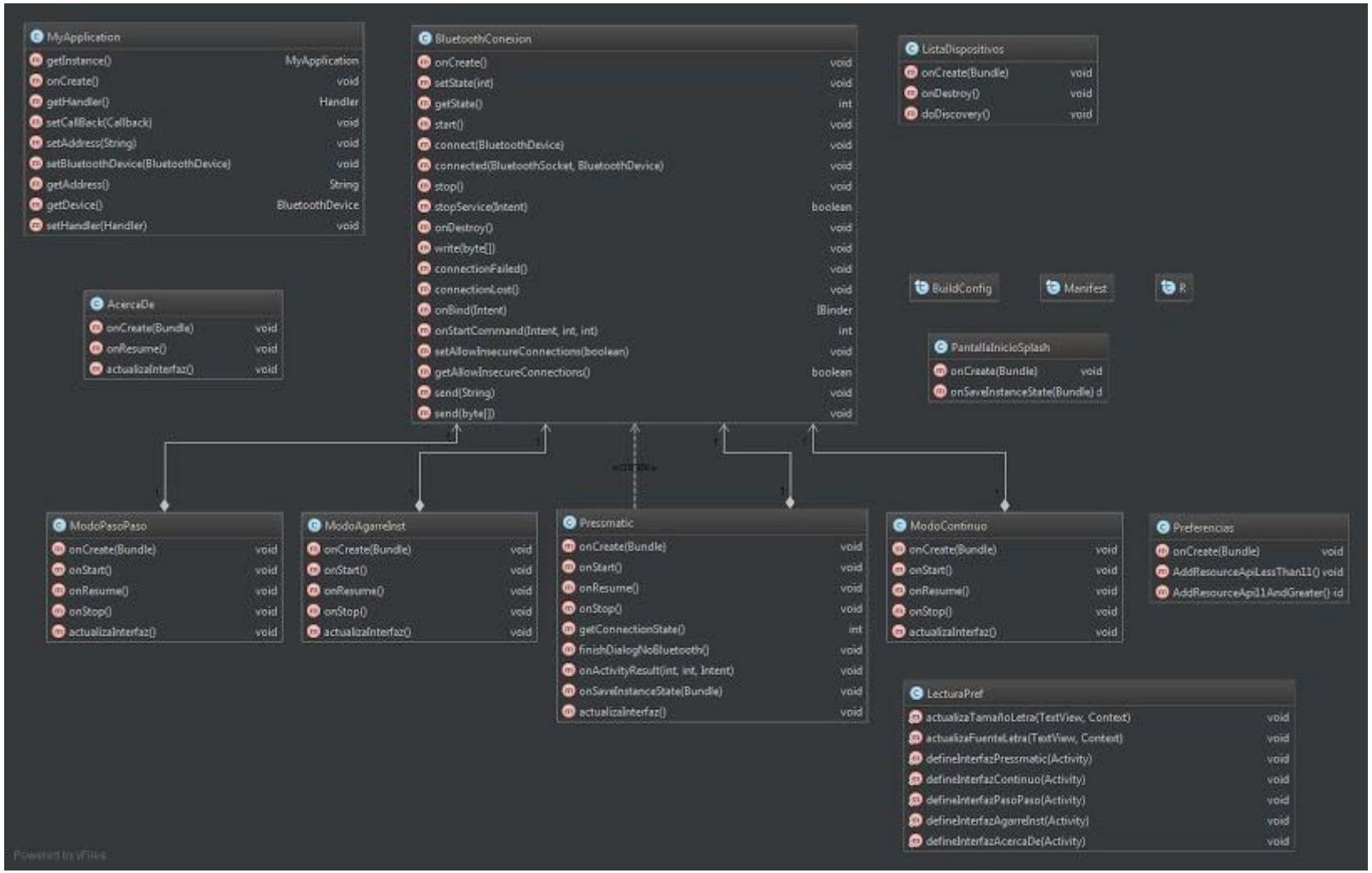

Figura 4.10 Diagrama de Clases de Aplicación PRESSSMATIC

4.2 Módulo de Control de voz para PRESSMATIC basado en EasyVR y Arduino.

Como ya se ha comentado con anterioridad, la principal finalidad de añadir una conexión Bluetooth al dispositivo reside en la capacidad de personalización y adaptación a diversas situaciones que ofrece mediante la posibilidad de comunicarse con una gran variedad de dispositivos. Una de las maneras de hacer cualquier dispositivo más accesible es el control por voz, sobre todo en casos más severos.

Este control de voz se puede obtener a través de varios servicios como los ofrecidos por Apple y Google en sus respectivos sistemas operativos móviles Android y iOS, pero requieren una conexión a internet. A fin de evitar la necesidad de una conexión a internet permanente para poder operar con el módulo y así eliminar restricciones, se optó por utilizar EasyVR.

EasyVR es un módulo robusto y barato que integra algoritmos de reconocimiento de voz y atenuación de ruidos. Además de hacer el procesamiento sin recurrir a enviar el sonido a terceros, tiene una sencilla integración en Arduino gracias a sus librerías y a que es compatible con el protocolo UART [25]. Sumando esto al sencillo interfaz de usuario utilizado para la configuración del módulo y a que soporta hasta 32 comandos personalizados, se convertía en la mejor solución de las disponibles en el mercado.

Para poder trabajar adecuadamente con EasyVR, es necesaria la utilización de un microprocesador subyacente. Para hacer el prototipo se escogió, por simplicidad y tamaño, el Arduino NANO con un procesador ATMega 328 junto con un módulo HC-05, el mismo que el utilizado en el propio dispositivo PRESSMATIC, además del Shield para Arduino de EasyVR. Estos 3 módulos se pueden integrar en un pequeño dispositivo sin la necesidad de demasiados ajustes a los circuitos junto con una pila y todos tienen un consumo de energía relativamente bajo.

Esto es debido a que el módulo EasyVR no puede mandar más datos al exterior que las identificaciones positivas en determinados grupos de comandos de voz. Esto quiere decir que necesita recibir órdenes y procesar los flags de éxito en el reconocimiento desde un microcontrolador subyacente para poder realizar acciones como cambio de grupo de órdenes o reproducir sonidos almacenados. Gracias a la librería de Arduino, es posible usar cualquier microcontrolador Arduino para hacer este control de una manera sencilla.

Para ello se deben haber configurado los drivers de Arduino y de la placa adecuada en el ordenador sobre el que se va a trabajar, además de configurar los puertos USB como puertos serie. Además son necesarias dos librerías a parte del conjunto de funciones estándar [26].

La librería de SoftwareSerial, permite crear un puerto serie a partir de los pines digitales del Arduino. Esto resulta necesario ya que se opta por trabajar con el Arduino Nano que solo dispone de uno y está destinado a la comunicación con el módulo Bluetooth. En segundo lugar está la librería de EasyVR, necesaria por las funciones de comunicación con el módulo y el anteriormente mencionado objeto EasyVRBridge.

En primer lugar hay que crear el objeto EasyVR que representa el módulo y al cual se pasa al ser construido el objeto un puerto de SoftwareSerial. Posteriormente se declaran los grupos mediante la palabra clave `enum`. En el `enum Groups` se ha de hacer referencia a todos los grupos y asignarles un entero a fin de hacer el código más fácil de entender. Posteriormente se

enumera individualmente cada comando en su grupo, asignándoles enteros diferentes y se crean dos variables `group` e `idx` del tipo `int8_t` que se encargarán de almacenar estos enteros.

En la aplicación, en la función `setup` , a modo de feedback para el usuario, se configura un puerto de salida para un LED que se ilumine cada vez que el dispositivo reconoce un comando. También se especifica que mientras no se detecte el EasyVR el LED parpadee rápidamente para informar al usuario de que algo sucede con el módulo. Por último se definen el Timeout time y el lenguaje estándar para comandos no dependientes de usuario. A pesar de que los comandos utilizados no son independientes es necesario definir un lenguaje. Por último se hace un setPinOutput para poner a nivel bajo el Pin IO1 de la EasyVR y se define el grupo de comienzo o TRIGGER, por definición siendo este el grupo 0.

Antes de inicializar el `void loop()`, equivalente a main en Arduino, se declara la función `action()`. Esta función es creada por el propio VRCommander tras generar los comandos y dar al menú File y a Generate Code. Esta función controla las transiciones entre grupos y las acciones realizadas al reconocer comandos.

Este montaje está basado en el shield de Arduino EasyVR y corresponde al siguiente esquema.

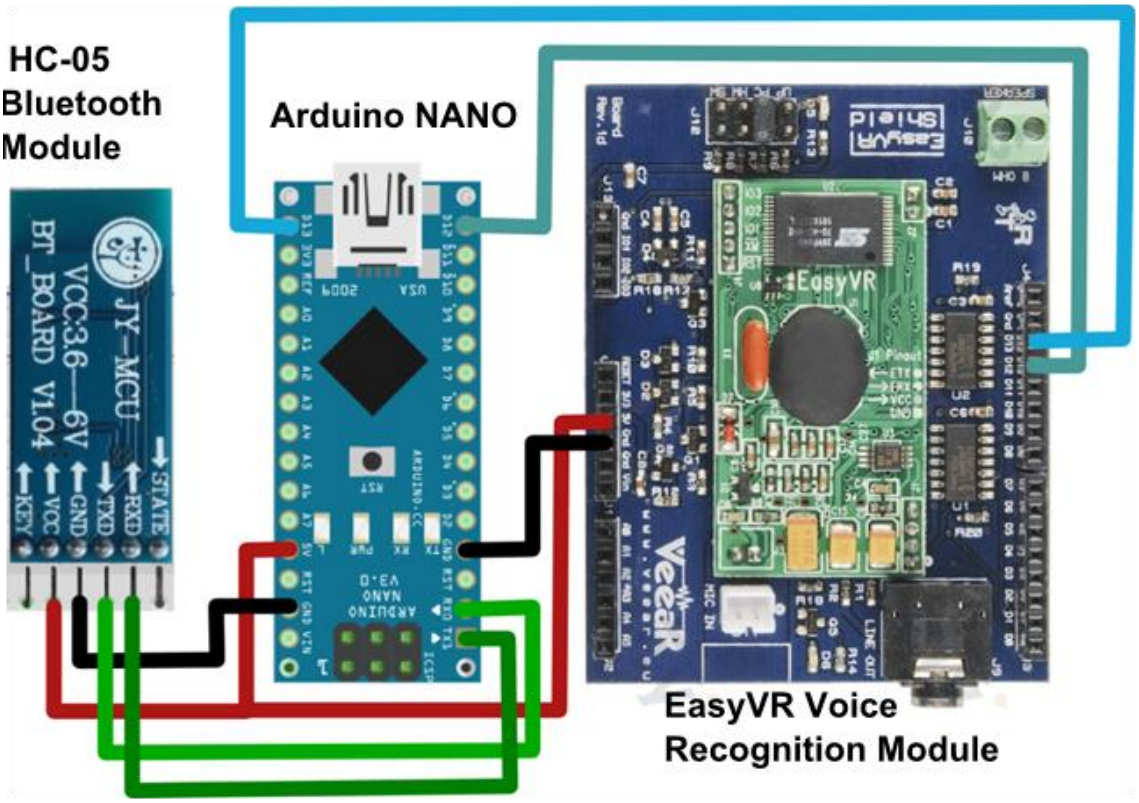

Figura 4.11 Esquemático del montaje de módulo de control basado en EasyVR

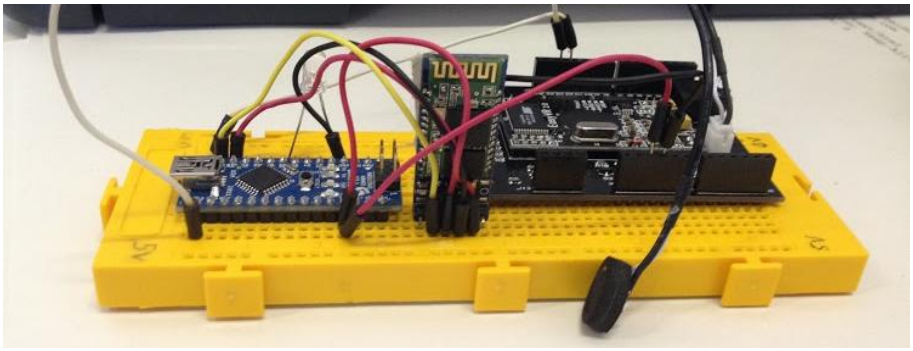

Figura 4.12 Montaje del Prototipo de control EasyVR

En el loop se activa el método del objeto EasyVR recognizeCommand(int8_t group) y el Arduino queda en espera hasta que se reconoce un comando.

```
easyvr.recognizeCommand(group);
```

Esto activa el EasyVR y hace que espere hasta que reciba un comando que pueda identificar. El Arduino espera en un ciclo do-while vacío hasta que EasyVR identifica el comando. Una vez se reconoce el comando, este se copia a `idx` y se comprueba si se ha reconocido adecuadamente el comando o si no ha sido un falso positivo.

```
idx = easyvr.getCommand();
  if (idx >= 0)
  {
    easyvr.playSound(0, EasyVR::VOL_FULL);//Plays a feedback sound if headphones
are connected
    action();                             //Takes actions in function of the
command recieved
    digitalWrite(ledPin, HIGH);           //Blinks the LED
    delay(200);
    digitalWrite(ledPin, LOW);
  }
```

Este código ejecuta 2 formas de informar al usuario del éxito del reconocimiento: un parpadeo de un LED y un pitido a través de la salida para auriculares integrada en el Shield de EasyVR.

Adicionalmente ejecuta la función `action()`. Esta función consiste de varios "switch-case". Uno principal que reconoce el grupo de comandos en el cual EasyVR se halla actualmente y un segundo anidado en cada caso del primer switch con los distintos comandos correspondientes a cada grupo.

Por ejemplo, si se observa el grupo 4 del código del módulo de control, correspondiente a la herramienta cortaúñas, se observa lo siguiente.

```
void action()
{
    switch (group)
    {
         ...
   case GROUP_4:                //Nail Clipper menu
```

```
      switch (idx)
      {
      case G4_CORTAR:            //cut once

        Serial.print('8');
        break;
      case G4_INICIO:            //Back to menu
        Serial.print('a');
        group = GROUP_1;
        break;
      }
      break;
         ...
}
```

En este código una identificación positiva del comando CORTAR manda por el puerto serie el comando correspondiente al módulo Bluetooth. En cambio, una identificación positiva del comando INICIO, además de enviar el comando correspondiente cambia el grupo asignado al Grupo 1, que es el que contiene los comandos de selección de herramienta. Esto permite al usuario circular a través de los distintos comandos realizando una emulación fidedigna de la pantalla integrada, a la par que facilita que no haya confusiones entre comandos.

En la edición actual del código, el proceso de funcionamiento se inicia mediante el comando de voz PRESSMATIC, que al ser una palabra tan reconocible impide que el dispositivo se inicialice sólo en un ambiente ruidoso. Posteriormente se accede al menú, que incluye las mismas opciones que la pantalla integrada en PRESSMATIC. Una de las grandes ventajas es que, como ya se explicó en la sección de Bluetooth, las pantallas de PRESSMATIC siguen los movimientos a través del menú de EasyVR, de manera que se puede saber si el dispositivo ha reconocido adecuadamente los comandos de voz de manera sencilla. Cada grupo de control a parte del principal y el de inicio incluye la palabra clave INICIO para retornar a la pantalla principal. Si, por último, se desea dejar el módulo en modo de espera, se utilizará la palabra clave DESCANSA desde el menú principal.

4.2.1Entrenamiento de comandos para EasyVR mediante VRCommander

Los comandos necesarios para realizar el control de PRESSMATIC ya se especificaron anteriormente y son un sencillo conjunto de letras que se envían a través de la conexión Bluetooth. También se explica cómo conectar dos módulos HC-05 utilizando comandos AT en la sección Conexión de dos Módulos HC-05. Asumiendo la conexión funcional de ambos dispositivos, lo único que requiere explicación es como grabar los comandos utilizando el VRCommander.

El VRCommander es un software gratuito diseñado por los creadores de EasyVR que sustituye al modo estándar de programación mediante comandos seriales enviados en UART. Este modo de configuración viene explicado en el manual de usuario y permite una manipulación más directa de los parámetros del módulo de reconocimiento de voz. Sin embargo, dado que se pretende ofrecer la posibilidad al usuario de personalizar los comandos y a que no se requiere un nivel muy profundo de manipulación del módulo en la aplicación, se optó por utilizar la interfaz gráfica ofrecida.

Cabe destacar que para el uso del VRCommander es necesario incluir el siguiente código en cualquier programa de arduino que vaya a soportar el uso de éste programa:

El objeto de la librería EasyVR

```
EasyVRBridge bridge;
```

Y el siguiente chequeo durante el setup:

```
//Checks that the if Bridge Mode is enabled in the VRCommander
#ifndef CDC_ENABLED
  // bridge mode?
  if (bridge.check())
  {
    cli();
    bridge.loop(0, 1, 12, 13);
  }
  // run normally
  Serial.begin(9600);
#else
  // bridge mode?
  if (bridge.check())
  {
    port.begin(9600);
    bridge.loop(port);
  }
#endif
```

Mediante el uso de este código se chequea si el ordenador está pidiendo desde el VRCommander que el Arduino entre en modo puente, retransmitiendo los comandos seriales que van desde el ordenador a el módulo EasyVR conectado, en este caso, a los pines 12 y 13.

Una vez añadido esto al código de Arduino, se procede a realizar la conexión desde el PC. Para ello, un puerto USB ha de haber sido habilitado como puerto serie y los controladores de Arduino instalados correctamente como se explica en la página web de Arduino.

Cumpliendo estas condiciones, la conexión se realiza mediante la segunda barra de herramientas del programa VRCommander seleccionando el puerto serie del ordenador al cual se halla conectado el Arduino en el menú desplegable situado a la izquierda de la ventana o haciendo clic en File y seleccionando la opción en el desplegable Port. Una vez elegido el puerto se pincha el botón situado a la izquierda del desplegable para conectar el módulo o se hace clic en File y se selecciona la opción Connect.

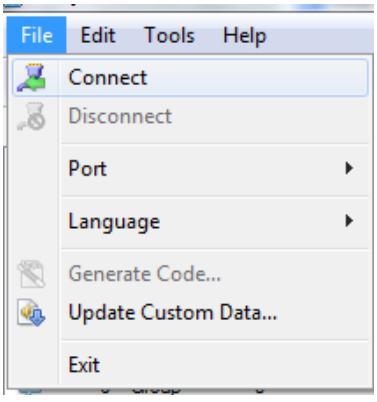

Figura 4.13 Menú desplegable File con opción Connect seleccionada de VRCommander

Posteriormente se procede a grabar los comandos de voz. Para iniciar la grabación de un comando de voz, se escoge el grupo objetivo en la columna situada a la izquierda y se añade el nuevo comando. Ésto se puede hacer mediante el botón de añadir comando de la barra de herramientas o en el menú Edit seleccionando Add Command.

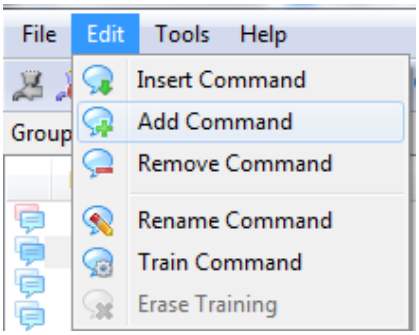

Figura 4.14 Menú desplegable Edit con opción Add Command seleccionada de VRCommander

Una vez se ha creado el comando, es necesario "entrenarlo" o realizar la grabación del comando para posteriores reconocimientos. Habiendo seleccionado el comando ya creado en la pantalla principal, se hace clic en Edit y se selecciona la opción Train Command.

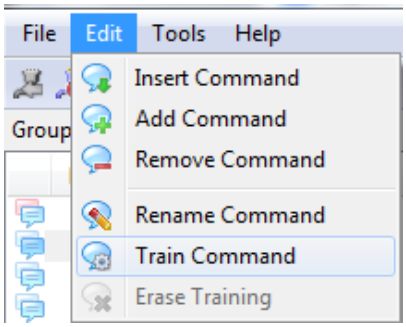

Figura 4.15 Menú desplegable Edit con opción Train Command seleccionada de VRCommander

Saldrá la siguiente ventana, en la cual se hará clic y se procederá a grabar el comando de voz situando el micrófono del módulo EasyVR a la distancia deseada.

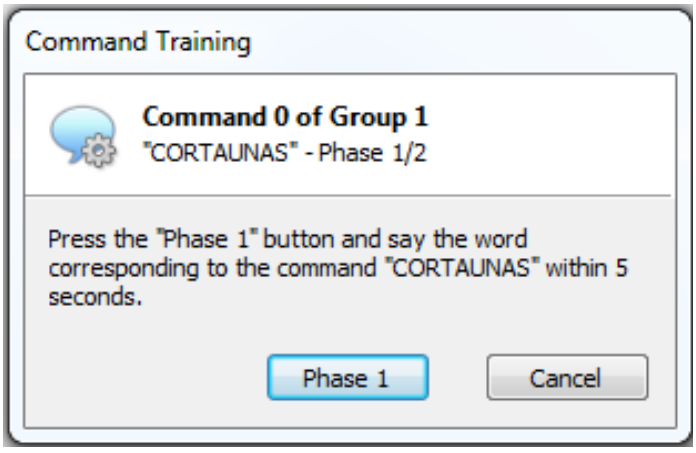

Figura 4.16 Ventana de entrenamiendo de comando VRCommander

El proceso de grabación se repite una segunda vez y si la grabación se realiza con éxito se puede proceder al siguiente comando. Es importante destacar que algunos comandos pueden provocar confusión con otros comandos cuando son parecidos entre sí. Para resolver esto el programa facilita una alerta que aparecerá en la columna Conflict de la pantalla principal, indicando también con que otro comando se está produciendo el conflicto.

Una vez se han creado los distintos comandos en los distintos grupos se puede chequear su integridad utilizando el botón de Testeo de grupos de la barra de herramientas o seleccionando el menú Tools y haciendo clic en Test Group

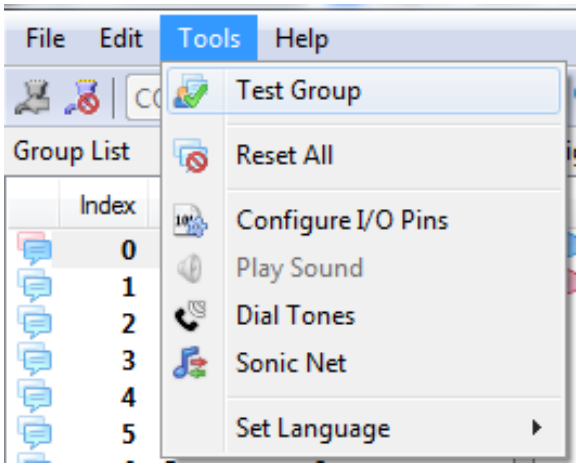

Figura 4.17 Menú desplegable Tools con opción Test Group seleccionada de VRCommander

Los comandos de voz necesarios para trabajar con PRESSMATIC están incluidos en la siguiente tabla y para que la compatibilidad con el programa de Arduino sea total deben estar tanto grabados en el mismo orden como presentar los mismos nombres y agrupaciones.

Así pues, los comandos a entrenar se agrupan en las siguientes categorías dentro del VRCommander.

Grupo	Comandos		
Grupo 0	PRESSMATIC		
Grupo 1	Cortauñas	Tijeras	Pinzas Grandes
	Pinzas Pequeñas	Bluetooth	Descansa
	Ayuda	Configuración	
Grupo 2	Abrir	Cerrar	Inicio
Grupo 3	Lento	Medio	Rápido
	Start	Stop	Inicio
Grupo 4	Cortar	Inicio	
Grupo 5	Desconectar	Inicio	

Tabla 4.1 Comandos PRESSMATIC para módulo EasyVR divididos por categorías

Capitulo 5

5. Diseño de la iconografía para interfaces PRESSMATIC en pantalla integrada y Android

El diseño de los iconos de Inkscape estuvo siempre supeditado a las normativas de accesibilidad vigentes y se realizó bajo supervisión de los desarrolladores de los interfaces gráficos Rodrigo Martín y Alejandro Vega [4] , [5].

El gran foco de atención fue la visibilidad de los iconos y la capacidad de reconocer su función de una manera sencilla visualmente. Esto es en gran parte responsabilidad del color utilizado. El color es una propiedad importante a tener en cuenta, dado que tiene que presentarse en combinaciones diseñadas de tal manera que los usuarios con problemas de visión puedan beneficiarse de los iconos. También es necesario que las combinaciones de colores produzcan el suficiente contraste para que se proporcione una diferenciación suficiente entre el primer plano que ocupan los iconos y el fondo.

El contraste se puede entender como una diferencia perceptible de brillo entre dos puntos de una imagen. Así pues, se estudió la generación de contraste mediante el uso de colores complementarios. Los colores complementarios, por naturaleza, generan un alto ratio de contraste entre ellos y su superposición o propia proximidad resulta llamativa a la vista, resaltando el brillo de ambos colores.

Figura 5.1Círculo Cromático

El contraste también se puede generar mediante una súbita caída o subida de brillo en el contorno del icono. A fin de asegurar que ambas posibilidades se contemplaban, se planteó conseguir, mediante un efecto de sombreado negro sumado a un efecto de brillo sutil sobre el icono en su esquina inferior izquierda dar alto contraste al icono a la par que fuera estéticamente agradable.

Sin embargo, el color no es el único aspecto a tener en cuenta. La normativa contempla además que la funcionalidad del botón sea clara y concisa. Muchos de los botones de PRESSMATIC

están pensados para mostrar un texto de alto contraste mediante el método de sombreado anteriormente descrito, aplicado al texto. Sin embargo, en algunos botones en los cuales un texto explicando la funcionalidad podía resultar excesivamente largo se recurrió a pictogramas que, en caso de duda, puede ser consultada su funcionalidad mediante el botón de ayuda.

Todas estas posibilidades se combinaron para generar los iconos de PRESSMATIC. Las siguientes imágenes reflejan la aplicación de estos principios en forma de Icono + Fondo para generar muestras de los diversos sets de Iconos.

Figura 5.2 Propuestas iniciales para combinaciones cromáticas de interfaz basadas en el círculo cromático

De entre estos sets, se decidió crear un set de iconos en Rojo #FF0000, que corresponde a un rojo puro en la escala Hexadecimal de color, con sombras en Negro puro #000000 y brillos en Blanco puro #FFFFFF.

Figura 5.3 Icono plano PRESSMATIC en rojo

Los sets de iconos debían incluir además una versión alargada de un ancho tres veces mayor que el icono estándar para acomodar las necesidades de botones que se pulsen con mayor frecuencia y una versión reducida de la mitad de alto para el botón de retroceso de la pantalla 4D.

Figura 5.4 Muestra de iconos alargados y reducidos

Es además necesaria una versión pulsada de cada icono. Para los iconos pulsados se decidió utilizar un tono de rojo más oscuro para dar la sensación de presión, para el cual se escogió el valor #D40000. Además se utilizó para la creación del marco un tono amarillo #FFFF00.

Figura 5.5 Icono plano PRESSMATIC en rojo presionado con marco amarillo

Para conocer el procedimiento de creación de un kit de iconos se recomienda consultar la guía de creación de iconos para PRESSMATIC incluida en el Anexo.

Además de los iconos en tonos de rojo se decidió crear un set de iconos azules a fin de cumplir con la normativa de accesibilidad al permitir personalizar la interfaz. Los únicos cambios realizados fueron a los colores de fondo, que en el caso del icono sin presionar corresponde al Azul #0088AA y en el caso del icono presionado al Azul #006680.

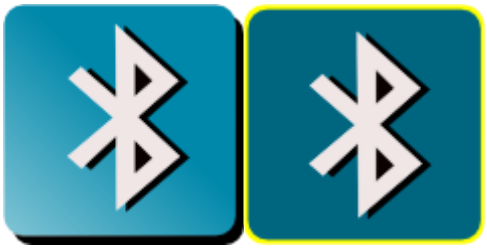

Figura 5.6 Ejemplo de Kit de Iconos Azul para PRESSMATIC. Botón Bluetooth sin presionar y presionado respectivamente

Así pues, con los dos sets de icono se cumplía la normativa necesaria en lo correspondiente estableciendo un diseño básico y sencillo de implementar, fácilmente ampliable en caso de que fuera necesario expandir los kits por aumentos en el número de herramientas o en la funcionalidad.

Capítulo 6

6. Pruebas para el protocolo Bluetooth

En este capítulo se recopilan las pruebas ejecutadas sobre los diversos módulos que forman parte del proyecto. El foco principal esta sobre el módulo PRESSMATIC y sobre la efectividad y estabilidad de su conexión tanto con móviles Android como con otros módulos HC-05.

Para hacer las pruebas sin necesidad del dispositivo PRESSMATIC al completo, se toma el propio módulo Bluetooth HC-05 contenido en el dispositivo y se conecta a, en el caso de las pruebas realizadas, una placa Arduino UNO, aunque también hay compatibilidad comprobada con Arduino NANO. El siguiente esquemático representa la conexión

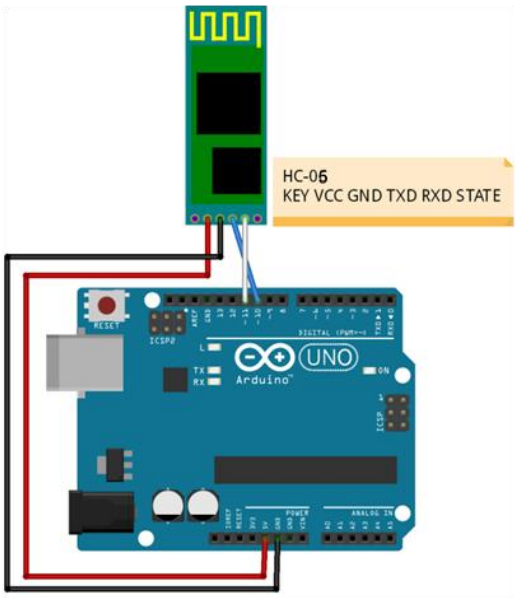

Figura 6.1 Conexión de módulo Bluetooth de PRESSMATIC a placa Arduino UNO

Una vez conectado, es necesario cargar al Arduino el código BT_ProofBench. Este código simplemente crea un puente entre la salida serial del módulo Bluetooth y el monitor serial del entorno Arduino, a fin de poder ver los resultados en la pantalla del ordenador.

Una vez preparado este banco de pruebas, se procede a realizar las comprobaciones de funcionamiento con el módulo EasyVR y la aplicación de Android.

6.1 Pruebas con aplicación Android

Durante el desarrollo del proyecto se ejecutaron varias pruebas con la aplicación PRESSMATIC a fin de ir determinando los posibles fallos y los posibles puntos a mejorar.

Las pruebas en etapas iniciales del desarrollo de la aplicación dieron resultados positivos para el modelo de aplicación con una sola actividad. Cada pulsación de diversos botones de la interfaz inicial mandaba por serial adecuadamente los caracteres del protocolo, lo cual trasladado a PRESSMATIC produciría las acciones deseadas.

Como se explicó en la sección 4.1.1 de este proyecto, ésto no resultaba suficiente para cumplir con las necesidades del proyecto, pues son necesarias varias pantallas para que los interfaces sean cómodos para el usuario. Así pues, se intento restaurar la conexión en cada cambio de actividad.

De nuevo, los resultados eran positivos. Los comandos llegaban adecuadamente al banco de pruebas. Sin embargo, el gran inconveniente, como ya se explico anteriormente en el apartado 4.1.1, era que cada cambio de actividad supone un nuevo intento de conexión con el proceso asociado. El total de tiempo del proceso, aun habiendo almacenado el módulo objetivo en la memoria del programa, era de 10 segundos aproximadamente, lo cual hacía la navegación torpe e incómoda.

Así pues, la última prueba, basada en la conexión como `service` detallada en la sección 4.1.2, crea una conexión estable, ágil y que devuelve resultados positivos al banco de pruebas.

Con una versión definitiva de la forma de conexión, posteriores pruebas tuvieron como objetivo comprobaciones en los cambios de interfaz y en retoques internos de la máquina de estados, manteniendo el protocolo invariante y por tanto devolviendo resultados idénticos a los anteriores.

6.2 Pruebas con módulo de control Por voz para PRESSMATIC basado en EasyVR

Las pruebas con EasyVR requirieron previamente de la configuración del módulo HC-05 integrado en el módulo de control de voz como maestro siguiendo las indicaciones descritas en el apartado 3.3 Comunicación Maestro-Esclavo entre dos módulos Bluetooth HC-05. Una vez configurado de esta manera, y cargado el código Bluetooth_Module_EasyVR se puede proceder a las pruebas.

Las pruebas realizadas con la primera implementación del módulo dieron resultados muy positivos. El reconocimiento de voz en un entorno estable y libre de ruidos otorgaba un alto grado de reconocimiento de los comandos. Debido a que EasyVR implementa algoritmos DTW de reconocimiento de voz, las pruebas en entornos ruidosos dieron peores resultados, con activaciones ocasionales debidas a conversaciones próximas al módulo.

Así pues, se observaron dos puntos de mejora. La primera versión del software acumulaba todos los comandos en un único grupo, con la palabra PRESSMATIC como clave de activación previa para utilizar cualquier comando. Esto reduce las posibilidades de activación autónoma debido a lo reconocible de la fonética de PRESSMATIC, pero resulta lento y tedioso, dificultando el movimiento a través de distintos modos de operación. La solución a esto consistió en implementar el actual esquema de operación de EasyVR, explicado en el apartado 4.2 Módulo de Control de voz para PRESSMATIC basado en EasyVR y Arduino.

Este esquema consiste en una emulación del sistema de pantallas implementado actualmente en la pantalla integrada de PRESSMATIC y en la aplicación Android. Evitando el uso continuado de la palabra clave PRESSMATIC otorga un alto grado de fluidez al modulo.

El problema del ruido, por desgracia, resulta inherente al algoritmo de EasyVR en lo referente a comandos personales. Es por ello que se procedió a grabar los comandos con una posición de micrófono más próxima a la fuente de sonido para así forzar a una mayor potencia del sonido para el reconocimiento del comando, con mejores resultados.

Siguiendo las indicaciones anteriores, en un entorno controlado con comandos grabados adecuadamente, la tasa de éxito en el reconocimiento de voz y de comandos supera el 80%.

Capítulo 7

7. Conclusiones y Líneas Futuras

7.1 Conclusiones

El objetivo de este proyecto era el desarrollo de un protocolo de comunicación sencillo y fiable que permitiera la comunicación a través de comunicación Bluetooth a fin de diversificar las posibilidades de control de PRESSMATIC, repercutiendo finalmente en el grado de adaptación del aparato al usuario final.

El desarrollo de cualquier protocolo de comunicación siempre viene supeditado a la aplicación que está por encima de él. Así pues, todo el proceso de desarrollo ha estado enfocado a seguir y respetar en todo momento el funcionamiento básico del dispositivo. (este protocolo se ha basado en las directrices y cambios que fueron sucediendo al)

El principal foco de desarrollo ha recaído sobre la tecnología Bluetooth, base de todo el proyecto. Está presente tanto en la integración en el dispositivo PRESSMATIC como en el control externo por voz a través de EasyVR y la aplicación móvil de PESSMATIC, permitiendo un alto grado de adaptación a las necesidades del usuario. Sin embargo, la manera más correcta en la que puede entender la tecnología Bluetooth en éste proyecto como un puente, por lo que no hay que excluir el papel representativo de otras tecnologías. El reconocimiento de voz y la tecnología móvil de los Smartphones y otros dispositivos Android son los que facilitan el objetivo primordial del proyecto, mejora del control y adaptación al usuario del dispositivo.

Es clara pues la razón por la que la tecnología Bluetooth fue escogida. Presente en prácticamente todo dispositivo Smartphone de la actualidad, sus seguros protocolos y la estabilidad de la conexión solo añaden valor y propósito a PRESSMATIC. A la comunicación rápida y eficaz se le suman su sencillez de integración y su precio reducido, lo cual representa una opción sumamente atractiva a incluir en un producto que, por definición, se pretende que sea asequible.

Todas estas tecnologías fueron respaldadas a lo largo del proyecto por tecnología basada en Arduino, el cual ha facilitado el desarrollo del proyecto gracias a su plataforma y lenguaje de programación sencillo de utilizar. La placas de desarrollo Arduino reducen en gran medida el tiempo de desarrollo y los chips ATMEGA resultan sencillos de integrar en los circuitos finales.

Gracias al uso de estas tecnologías abiertas, es posible concluir que se han alcanzado con éxito los objetivos de este proyecto al permitir, en base a él, el desarrollo de muchas posibles aplicaciones futuras.

7.2 Líneas Futuras

El proyecto en su nivel de desarrollo actual cumple con las expectativas y objetivos sentados en un principio. Sin embargo, siempre es posible añadir nuevas funcionalidades y realizar mejoras a las ya presentes.

Permitir que las aplicaciones externas sigan las acciones de la pantalla integrada una vez conectadas. Las acciones de control externas enviadas a través de Bluetooth al dispositivo PRESSMATIC provocan actualmente un seguimiento por parte de la pantalla. Resultaría beneficioso que el proceso inverso ocurriera, dándole más libertad al usuario de elegir que medio utilizar para dar órdenes al dispositivo sin que ocurra una pérdida de sincronía.

Implementar un módulo compatible con IPhone y una aplicación móvil. La tecnología actualmente implementada no es compatible con los productos móviles de Apple, lo cual limita las capacidades del dispositivo. En etapas iniciales se desestimó enfocar el desarrollo de Bluetooth hacia productos Apple, en gran medida porque la cuota de mercado de Android supera el 79% del total de dispositivos, pero incluir estos productos en el rango de dispositivos compatibles sólo resulta positivo. Este desarrollo enfocado hacia compatibilidad con IOS requeriría además del desarrollo de una aplicación dedicada.

Para cubrir todos los ámbitos del mercado de dispositivos móviles también existe la posibilidad de **crear una aplicación para el OS Windows Phone.**

Crear un programa de ordenador de control por voz. En base las diversas librerías de software libre disponibles sería posible crear un programa similar al utilizado en EasyVR que permitiera el uso de la potencia de procesado de un ordenador para tareas de reconocimiento de voz. Ésto ampliaría aun más las capacidades de PRESSMATIC dado que se podrían aprovechar las capacidades de ordenadores ya configurados para ser accesibles a fin de asistir al usuario.

Integración de controles en otros dispositivos de asistencia. La comunicación Bluetooth abre posibilidades para crear módulos de control que se puedan integrar en sillas de ruedas y en otros dispositivos de asistencia sin gran dificultad.

Capítulo 8

8. Presupuesto y Planificación

En este capítulo final se realiza un análisis del presupuesto necesario para realizar el proyecto, incurriendo tanto en gastos de materiales utilizados como de trabajo personal.

8.1 Presupuesto

El presupuesto necesario para realizar el proyecto se desglosa en varios conceptos como los gastos por materiales, gastos personales y otros costos directos.

En las próximas tablas se muestra el presupuesto completo, una ficha con todos los conceptos anteriormente mencionados. Sus valores han sido calculados teniendo en cuenta los siguientes datos.

Costes de Recursos Humanos: Estos costes son los generados por las personas que participan y desarrollan el proyecto, detallando sus horas de trabajo. Para realizar este proyecto ha sido necesaria la participación de un Graduado en Tecnologías Industriales dedicando al proyecto un tiempo aproximado de 6 meses. Si se retiran del cálculo los fines de semana, se obtiene una media de 20 días laborables mensuales, resultando en un total de 120 días trabajados. Asumiendo una jornada laboral de 5 horas diarias se obtiene un total de 600 horas trabajadas. Teniendo en cuenta el dato de la medida hombre mes como 131.25 horas se obtiene, como se muestra en la tabla 8.1,un total de **11.960€**.

Costes de materiales y equipos: Estos costes engloban los gastos referentes a materiales utilizados para la realización del proyecto. En ellos se incluye el hardware y también otros servicios utilizados como internet. Para los diversos cálculos se asume una depreciación de 60 meses, excluyendo al uso de internet que mensualmente se renueva. Así pues, el total de estos costos, como queda reflejado en la tabla 8.2, es de **245,15€.**

Costes de Software: En esta sección de costes se incluyen los diversos software utilizados para el proyecto, así como el coste de los mismos. Si bien es cierto que los software no sufren una depreciación per se al ser las licencias adquiridas de duración ilimitada, se ha estimada que el tiempo en el que estos software quedan obsoletos es de 5 años o 60 meses. Teniendo en cuenta estos datos, el coste total por software, como se muestra en la tabla 8.3, es de **27€**.

Costes directos: Estos engloban el material de oficina utilizado en el proyecto, tal como se muestra en la tabla 8.4, por un total de **50€**.

La suma total de estos costes da un valor resultante de **14733.18 €**, justificados a continuación.

Costes de Recursos Humanos

Nombre	Categoría	Dedicación (hombre mes)*	Costes (hombre mes)	Coste
Alonso Rosado García	Graduado en Tecnologías Industriales	4,6	2600	11960
			TOTAL	11960

Tabla 8.1Tabla de Costes de Recursos Humanos

*1 Hombre mes equivale a 131.25 horas, siendo el máximo anual de dedicación unos 12 hombres mes, equivalentes a 1575 horas. Para un PDI de la Universidad Carlos III de Madrid, este máximo difiere y se sitúa en 8,8 hombres mes, equivalentes a 1155 horas.

Para los cálculos de amortización de software y hardware presentados a continuación se utiliza la siguiente fórmula.

$$Valor\ \ de\ Amortización = \frac{A}{B}\ x\ C\ x\ D$$

Donde:

A = nº de meses desde la fecha de facturación en que el equipo es utilizado.

B = periodo de depreciación (generalmente 60 meses)

C = coste del equipo sin IVA

D = porcentaje de uso que se dedica al proyecto

Costes de Equipo, Hardware y Servicios

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste Imputable
Ordenador Portatil	800	75	6	60	60
Samsung Galaxy S Advance	300	100	6	60	30
Arduino UNO	20	100	6	60	2
Arduino NANO	33	100	6	60	3,3
2 Módulos Bluetooth HC-05	9	100	6	60	0,9
Internet	49	50	6	1	147
EasyVR	39	100	3	60	1,95
				TOTAL	245,15

Tabla 8.2 Tabla de Costes de Equipo, Hardware y Servicios

Costes de Software

Descripción	Coste licencia (€)	Duración de la licencia	Tiempo de uso en el proyecto (meses)	Coste imputable (€)
S.O. Windows 7	90	Indefinida	6	9
Paquete Office 2010 Home and Student	120	Indefinida	4	18
IntelliJ IDEA Community Edition	0	Indefinida	6	0
Plataforma Android 2.3.3 (SDK)	0	Indefinida	6	0
Inkscape	0	Indefinida	4	0
			TOTAL	27

Tabla 8.3 Tabla de Costes de Software

Por último, otros costes directos asociados a materiales fungibles como papelería y cartuchos de tinta de impresora se agrupan en la siguiente tabla.

Otros costes directos	
Descripción	**Coste (€)**
Material de oficina	50
TOTAL	50

Tabla 8.4 Tabla de otros Costes Directos

Así pues, estos costes se agrupan resultando en la siguiente tabla

Costes Totales	
Concepto	**Coste(€)**
Personal	11960
Amortización	245,15
Otros costes directos	50
Costes indirectos	2451,03
Costes de software	27
TOTAL	**14733,18**

Tabla 8.5 Tabla de Costes Totales

Anexo

Esta guía tiene como objetivo aclarar el proceso detrás del desarrollo de los iconos de PRESSMATIC para expandir los conjuntos incorporados en este proyecto o crear unos completamente nuevos mediante el uso de la herramienta Open Source Inkscape.

El desarrollo de un kit de iconos consta dos partes básicas diferenciadas, el desarrollo del fondo del icono y el desarrollo del símbolo o texto incorporado encima de éste.

En el proceso de desarrollo del kit, se han de realizar seis fondos. Dos grupos de tres fondos diferentes correspondientes a los diferentes tamaños utilizados por PRESSMATIC, uno de los grupos presionado y el otro sin presionar.

Los fondos de icono son la base de todos los iconos de PRESSMATIC y aquello que determina gran parte del estilo de la iconografía del conjunto. Para este tutorial se utilizará una forma cuadrada de esquinas redondeadas y con sombras y brillo y una forma presionada de color más oscuro, sin sombras ni brillo y con marco de color como la utilizada en el actual set de iconos PRESSMATIC. También se especificará la creación de los iconos grandes y pequeños a través de la modificación de estos modelos básicos.

A.1 Creación de fondos de Icono para PRESSMATIC

Figura A.1 Iconos PRESSMATIC planos de kit rojo

Esta es la forma básica de un icono de PRESSMATIC de tamaño medio en sus dos estados: sin pulsar y pulsado respectivamente. Gracias al sistema de capas que incorpora Inkscape solo es necesario crear esta forma básica una vez, pudiendo así superponer la iconografía y la tipografía sin mayores complicaciones y sin riesgo de modificar involuntariamente la plantilla.

A.1.1 Icono sin Pulsar

Para empezar a construir un icono del estilo PRESSMATIC se selecciona en la barra de herramientas vertical izquierda la herramienta de cuadrado.

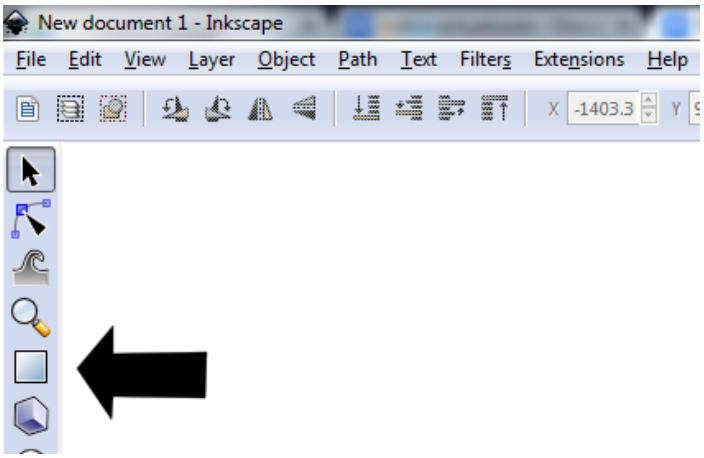

Figura A.2 Herramienta de Cuadro

Una vez seleccionada, creamos una forma de cualquier tamaño cliqueando en cualquier punto de la pantalla y arrastrando. No es necesario hacerlo preciso, dado que los ajustes se realizarán introduciendo valores numéricos en la barra de herramientas superior.

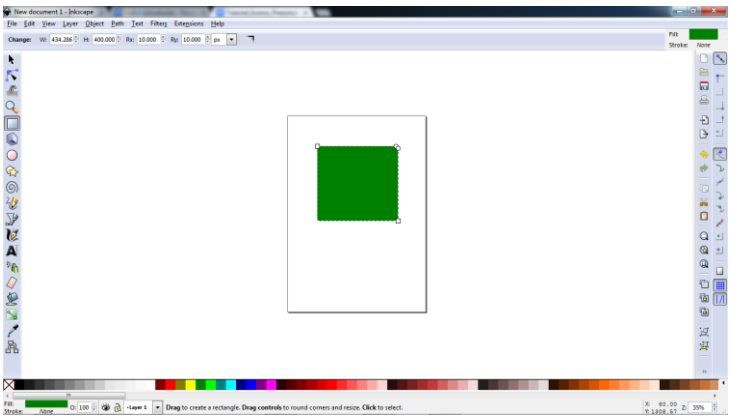

Figura A.3 Cuadro de tamaño aleatorio sobre la pantalla Inkscape

Dependiendo del tipo de interfaz que queramos hacer, se modificarán estos valores de una manera u otra. Para un interfaz al estilo de Windows Phone se eliminaría el redondeado y se utilizarían colores planos sin sombra. En este caso, se desea hacer un interfaz parecido al de IOS, con esquinas redondeadas y sombras, lo cual requiere un poco más de trabajo.

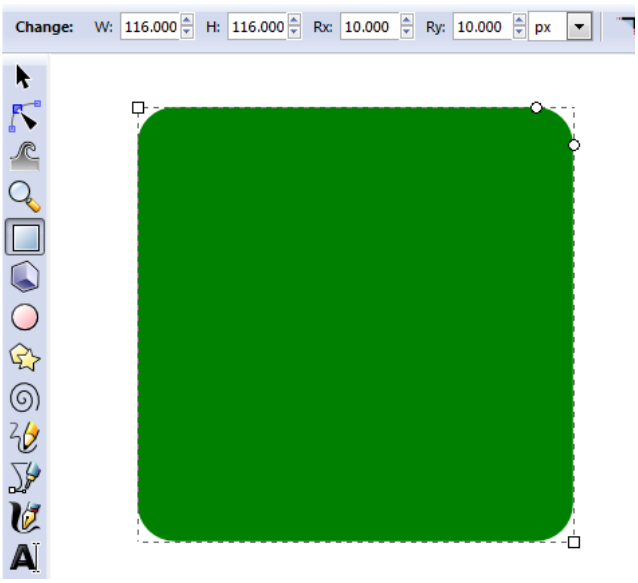

Figura A.4 Cuadro de color de dimensiones ajustadas para icono sin pulsar

Aunque los iconos van a ser de un tamaño de 120x120 px, se hace el tamaño de este cuadrado de 116x116 a fin de dejar espacio para el sombreado.

Ahora se abre la herramienta de capas abriendo la pestaña Layer haciendo clic en Layers o utilizando el comando Shift+Ctrl+L y aparecerá la siguiente columna en el lado derecho de la pantalla.

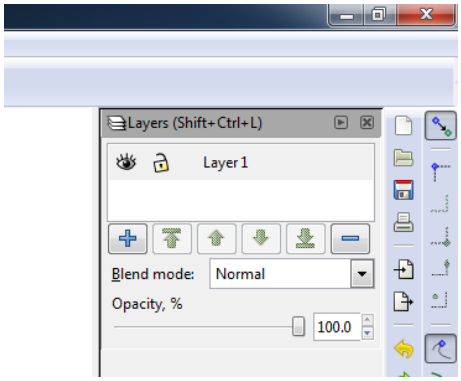

Figura A.5 Ventana de Capas

El símbolo del ojo indica la visibilidad de la capa y el del candado indica si la capa puede ser modificada. La capa original tiene por defecto el nombre Layer 1. Si se hace doble clic sobre el nombre, se permite modificarlo. Así pues, para facilitar el reconocimiento de cada capa y los elementos que contiene, se llama a esta capa Color_Icon_Medium_NoPress.

A continuación se hace clic en el símbolo de mas y se obtendrá la siguiente ventana. Dado que ahora se desea crear la capa que contendrá el sombreado del icono, creamos la capa por debajo del icono actual utilizando Below Current.

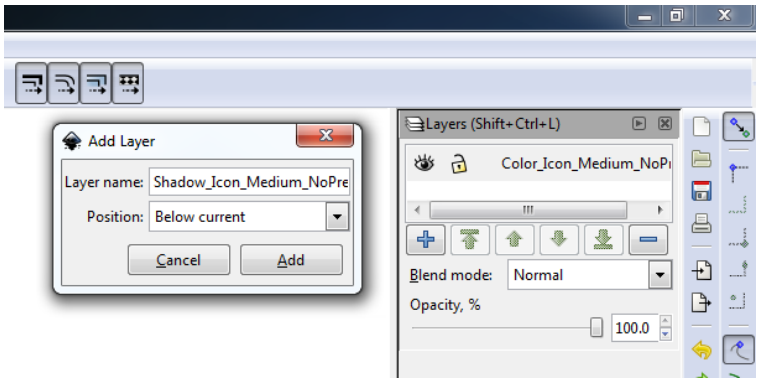

Figura A.6 Ventana de añadido de capas

Con los mismos pasos, manteniendo la selección en la capa color, creamos una nueva capa que contendrá el brillo por encima de la capa de color usando la opción Above current.

A continuación se copia el cuadro para hacer el brillo y la sombra. Esto se consigue haciendo clic izquierdo sobre el objeto y seleccionando copy y posteriormente paste seleccionando la capa creada correspondiente o usando el comando ctrl+c y luego el ctrl+v.

Una vez tenemos los tres cuadrados sobre la pantalla seleccionamos uno de los nuevos cuadros y lo pintamos de negro para hacer la sombra.

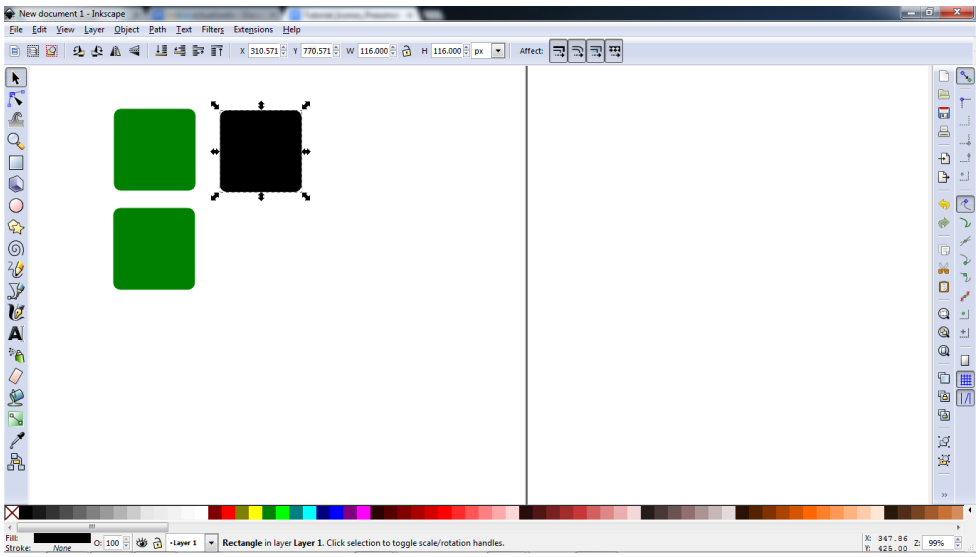

Figura A.7 Copia y cambio de color del cuadro inicial

Este cuadro se desplaza posteriormente debajo del verde inicial sumando 4 a su coordenada X y restando 4 a su coordenada Y para obtener así una figura que en total ocupará 120x120px.

Figura A.8 Cuadro de color sombreado de dimensiones 120x120 px

Una vez hecho eso, para evitar modificaciones involuntarias al contenido del icono y el sombreado, se bloquean ambas capas pulsando el símbolo del candado.

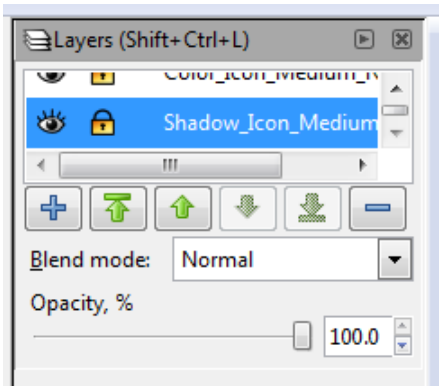

Figura A.9 Bloqueo de capa

Ahora solo queda añadir el brillo. Para ello, se selecciona el cuadro restante localizado en la capa Bright y se hace clic derecho sobre él. En el menú desplegable se selecciona la opción Fill and Stroke. Aparecerá un nuevo menú a la derecha de la pantalla con varias pestañas de las cuales se seleccionará la pestaña Fill, si no viene seleccionada por defecto. En esta pestaña aparecen varios cuadros con tipos de relleno, de los cuales se selecciona Linear Gradient.

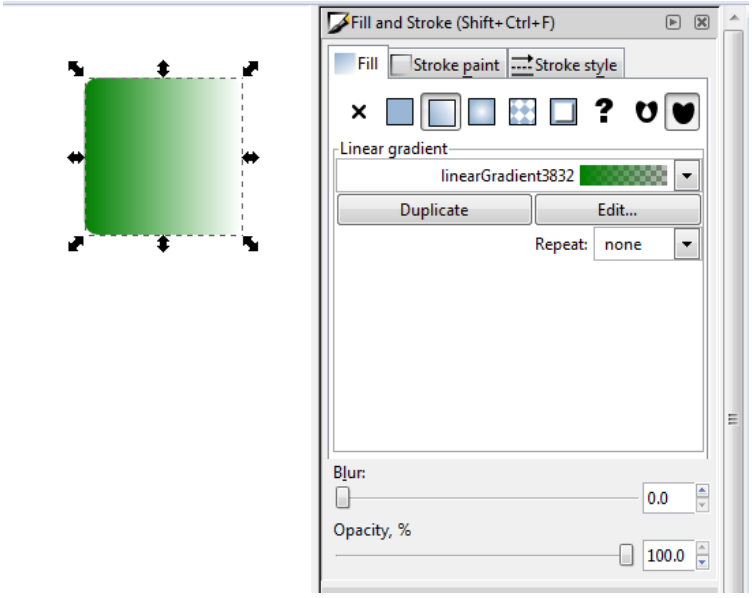

Figura A.10 Ventana de relleno y borde, configurada para obtener una gradiente de color

A fin de localizar sencillamente el cuadro, todavía no se cambia el color de la gradiente a blanco.

Por razones estéticas se decide que el brillo progrese desde la esquina inferior izquierda. El gradiente estándar no cumple ese requisito por lo que para modificarlo se necesario hacer doble clic sobre el cuadro. Aparece la siguiente imagen, representando el segmento del medio la gradiente de color como tal.

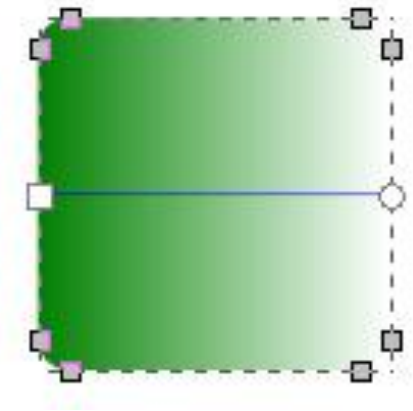

Figura A.11 Gradiente de color verde en pendiente horizontal

Modificando la posición de los extremos de este segmento y colocándolos en la esquina superior derecha e inferior izquierda se obtiene el efecto deseado.

Figura A.12 Gradiente de color verde en pendiente

Ahora se desplaza este cuadro a la misma posición X e Y que la del cuadro original y se cambia el color a blanco. El cambio de color se realiza haciendo doble clic sobre el punto del segmento de gradiente localizado en la esquina inferior izquierda, obteniendo así el siguiente resultado final.

Figura A.13 Resultado final. Icono plano sin pulsar.

A.1.2 Icono Pulsado

Para facilitar la creación de los kits de iconos es altamente recomendable el construir componente de color del icono pulsado en la misma posición que ocupa el no pulsado.

Para el icono pulsado se crean 2 capas nuevas: una para el color del icono presionado, más oscuro que el de sin presionar, y otra para el marco, colocada debajo de la de color. Se desbloquea la capa de color del icono sin pulsar y se copia el cuadro a la capa de color del presionado, ahorrando así la necesidad de repetir el proceso anterior, y se vuelve a bloquear.

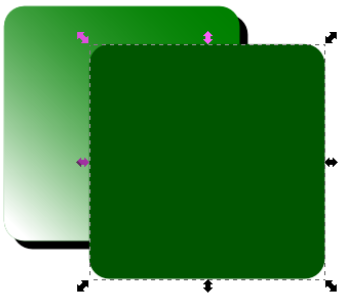

Figura A.14 Copia del cuadro de color

Este nuevo cuadro utiliza un tono más oscuro para dar la sensación de presión. Se posiciona este recuadro de color exactamente sobre el anterior y se cambia de capa.

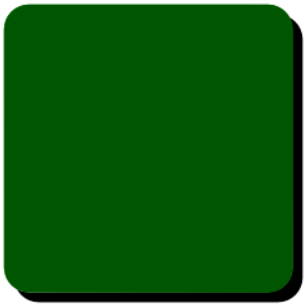

Figura A.15 Cuadro de color de icono presionado posicionado sobre cuadro de color de icono sin presionar.

Ahora se bloquean todas las capas menos la capa Frame y se ocultan todas las del icono sin presionar pulsando el icono de ojo en el menú de Layer.

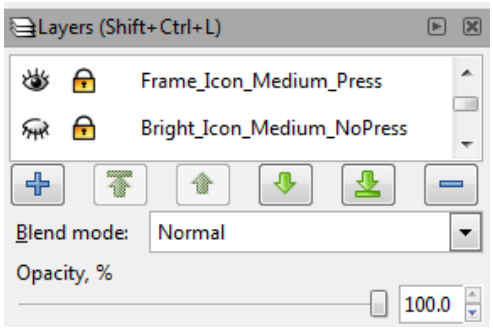

Figura A.16 Menu de capas. Ocultando el icono sin presionar.

En la capa del marco se copia un cuadro nuevo, cuyas dimensiones se cambian a 120X120 px y su color se cambia a un tono de color muy llamativo. Este marco se posiciona 2 pixeles menos en ambas coordenadas X e Y, dando lugar al siguiente resultado.

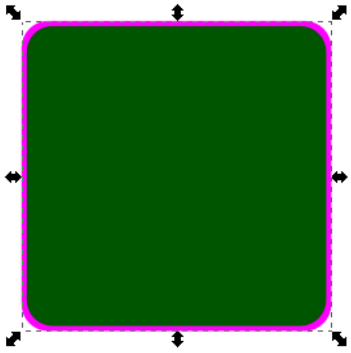

Figura A.17 Icono presionado final.

A.1.3 Ampliando y reduciendo tamaños

Inkscape permite la edición proporcional de grupos de objetos, lo cual, a la hora de crear nuevos tamaños de iconos, resulta extremadamente útil. Sin embargo, no es posible únicamente realizar esta edición, sino que hay que hacer unos ligeros ajustes. Esto permite crear los componentes restantes del kit de iconos de PRESSMATIC con relativa facilidad

Se crea una nueva capa llamada Icon_Large_Frame , en la cual se crea una copia de los cuadros del icono pulsado. Manteniendo la selección de todos los componentes se pulsa Ctrl+G para agrupar los objetos o se abre el menú desplegable Object y se selecciona la opción Group.

Una vez hecho esto, se va a la barra de herramientas superior y se modifica el valor de Ancho o Width, correspondiente a la letra W, del grupo y se multiplica por 3, alcanzando el valor de 360 px.

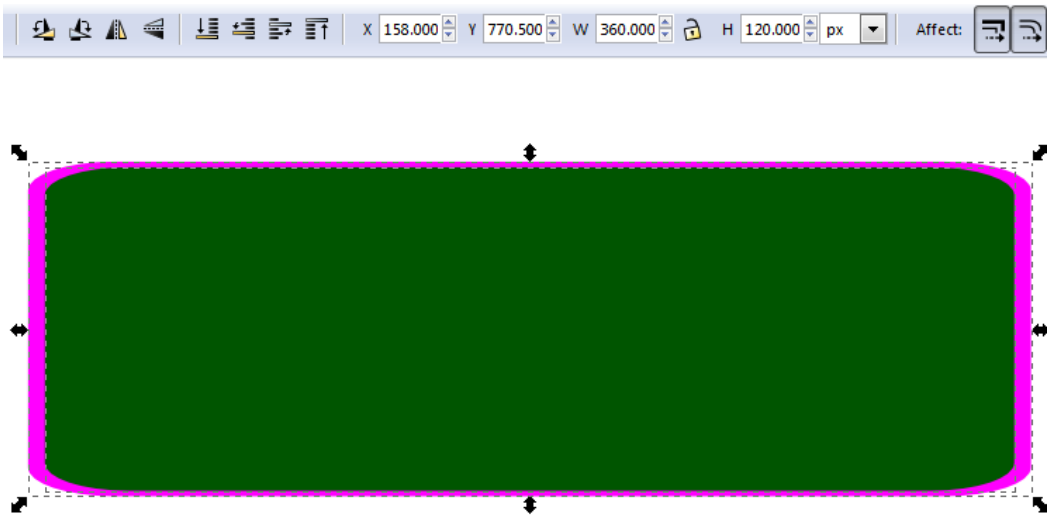

Figura A.18 Icono pulsado modificado

Como se puede observar, el redondeo de las esquinas se deforma al realizar la edición proporcional, por lo cual se ha de restaurar a sus valores de radio de 10px y 10px seleccionando cada uno de los componentes individualmente y cambiando su valor en la barra de herramientas superior. Una vez realizado el cambio, se dispone de un nuevo fondo básico sobre el que trabajar.

Figura A.19 Icono definitivo alargado pulsado

Esta técnica es la misma utilizada en la creación de todos los demás fondos básicos, ya sean de tamaño 120x60px o 360x120px y para las versiones pulsadas y sin pulsar.

Es importante destacar que en las versiones con brillo es posible que sea necesario cambiar la pendiente de la gradiente para que el brillo no incremente demasiado en intensidad.

A.1.4 Incluyendo Tipografía

Si se desea crear un icono que contenga un texto o un dibujo que incluya letras, Inkscape facilita las herramientas para incorporar el texto con una gran variedad de fuentes.

Se comienza creando una nueva capa que contendrá el texto, que en este caso, será PRESSMATIC. Esta capa debe posicionarse por encima de la base para que sea visible. Una vez seleccionada la capa, en la barra de herramientas izquierda se selecciona el siguiente icono.

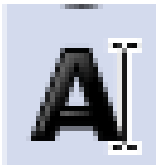

Figura A.20 Icono de Herramienta de Textos

Esta es la herramienta de textos. Una vez seleccionada, se picha y arrastra, creando el área sobre el cual aparecerá el texto deseado. Para este ejemplo se utilizará el texto PRESSMATIC.

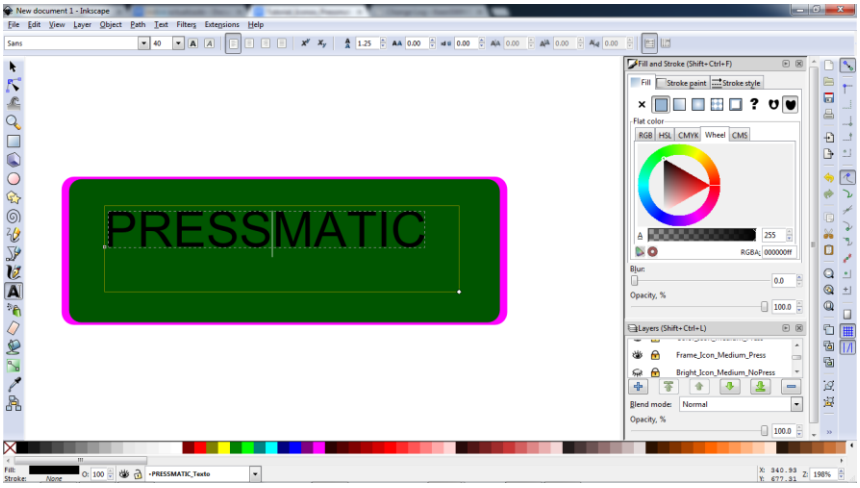

Figura A.21 Cuadro de texto sobre icono plano pulsado

Este texto está escrito en la tipografía estándar Sans en tamaño 40. Varios atributos pueden ser modificados en la barra de herramientas superior, y en el lateral derecho se observa que se nos da la posibilidad de cambiar de color o de crear gradientes con las letras. Para este ejemplo solo se utilizará el relleno tradicional y los colores negro y blanco, aunque las posibilidades son ilimitadas.

Para añadir un poco más de estilo al icono, se cambia el tamaño de letra a 50 para que se ajuste más al marco y se cambia la tipografía a Stencil. A su vez, se cambia el espaciado entre letras, también en la barra superior, a -4, llevando al siguiente resultado.

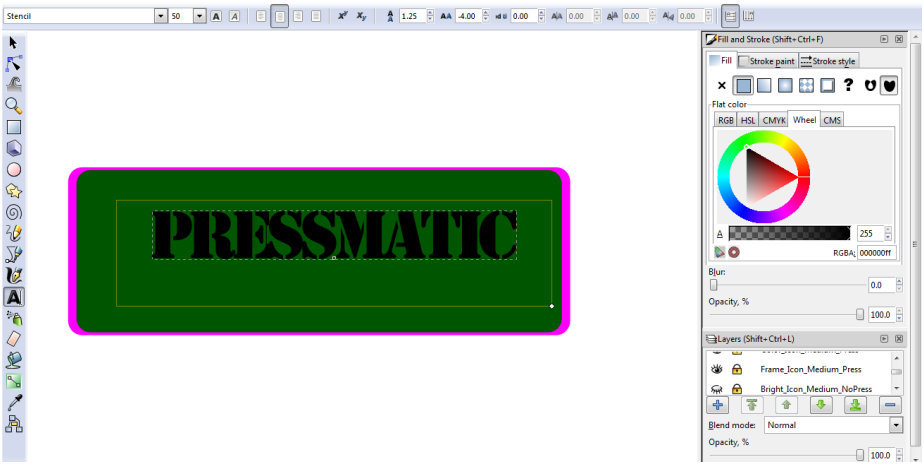

Figura A.22 Cambio de tipografia y de espaciado entre letras

A fin de dar el efecto de sombreado, se puede copiar el cuadro de letras y cambiarlo a color blanco simplemente usando la barra inferior de colores. Se recomienda hacer esto en la misma capa pues resulta más cómodo posteriormente a la hora de ocultarlo o exponerlo. Una vez copiado, se coloca al gusto, pues el posicionamiento más bello estéticamente depende de la fuente, y se agrupa y centra.

Figura A.23 Resultado Final

A.1.5 Añadiendo Imágenes

En muchos casos, la manera más efectiva y de mejor estética en la que se puede comunicar la intención de un botón es mediante un pictograma. Sin embargo, es muy posible que las imágenes disponibles no se ajusten al estilo deseado. Es por ello que puede ser necesario crear una propia.

En este caso de ejemplo, esta imagen de unas tenazas se ha elegido por su forma, pero sin embargo no puede ser utilizada porque su fondo blanco no tiene transparencia y porque además no sigue el estilo en blanco y negro deseado para los pictogramas de PRESSMATIC.

Figura A.24 Imagen de muestra, base para icono PRESSMATIC

Por ello se recurre a la herramienta de pluma. Esta herramienta permite crear formas complejas mediante unos pocos puntos utilizando el método conocido como dibujo vectorial.

Primero es recomendable incluir la imagen utilizada como plantilla en una capa aparte, que luego será borrada, y bloquearla. Una vez hecho eso, se crea una nueva capa, en este caso Pinzas_PRESSMATIC por ejemplo, y se selecciona la herramienta de pluma de la barra izquierda de herramientas.

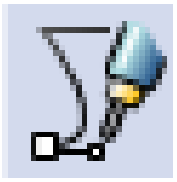

Figura A.25 Icono de herramienta de pluma

Esta herramienta permite generar puntos que serán unidos por líneas rectas si se selecciona la localización con un simple clic, o si se mantiene pulsado y se arrastra generarán líneas curvas con pesos modificables.

Figura A.26 Dibujo de una linea curva sobre la imagen de referencia.

Estos pesos determinan la curvatura de la línea, y se pueden modificar individualmente utilizando la herramienta de selección de puntos.

Figura A.27 Icono de herramienta de selección de puntos

Una vez se cierra el contorno, se obtiene el siguiente perfil.

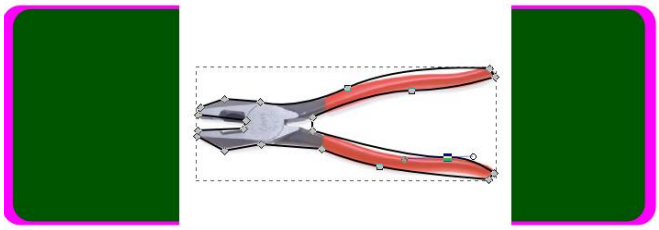

Figura A.28 Perfil cerrado, diferenciado como un objeto en el entorno de Inkscape

Este perfil cerrado es ya de por sí una figura como la del cuadro, que puede ser rellenada de color, utilizar gradientes, etc. En éste caso, a fin de seguir con la estetica del conjunto, se utiliza un relleno blanco y se retira el enmarcado, se copia la figura y se pinta de negro. Para posicionar el objeto copiado debajo del original, pero dentro de la misma capa, se utiliza el menú Objetc y la opción Lower. Finalmente se obtiene el siguiente resultado.

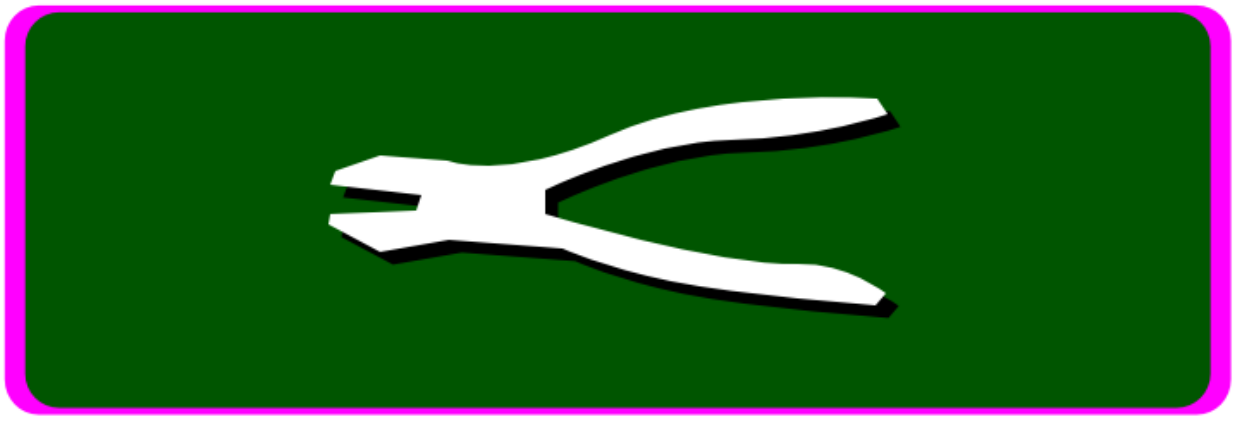

Figura A.29. Resultado final

A.1.6 Guardando los iconos

Lo único restante es guardar los iconos. A fin de que los iconos sean compatibles en el mayor número de dispositivos, lo mejor es exportarlos como bitmaps en formato PNG.

Para ello, primero, se selecciona el icono en su totalidad, es decir, se desbloquea la plantilla y se selecciona al completo. Posteriormente se va al menú File y se selecciona la opción Export bitmap o utilizar el comando Shift+Ctrl+E. Aparecerá la siguiente ventana.

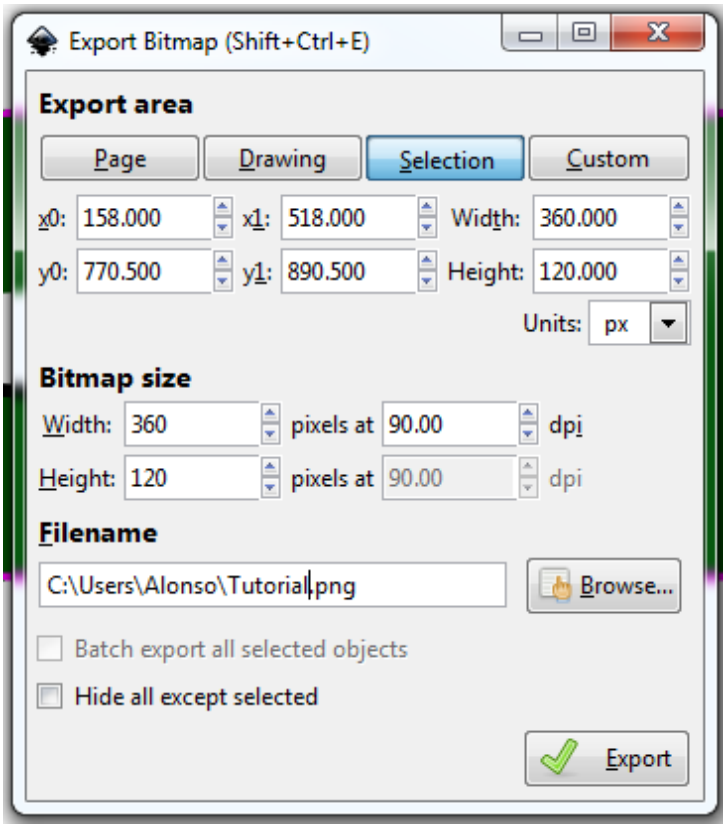

Figura A.30 Ventana de exportacion de mapas de bits.

En esta ventana se elige Selection bajo el título de Export Area y, una vez se ha decidido la carpeta de almacenamiento y el nombre del archivo, se selecciona Export.

Si se han seguido los consejos de esta guía, guardar un nuevo icono debería ser tan fácil como ir ocultando las distintas capas base y de iconos según sea necesario y cambiar el nombre con el que se desea exportar el archivo.

Glosario

GHz	Gigahertzio
WPAN	Wireless Personal Area Network
RF	Radiofrecuencia
SIG	Special Interest Group
EDR	Enhanced Data Rate
ATT	Attribute Protocol
GATT	Generic Attribute Protocol
LMP	Link Manager Protocol
L2CAP	Logical Link Control Adaptation Protocol
OBEX	Objects Exchange Protocol
Mbps	Megabit por segundo
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
SDP	Service Discovery Protocol
RFCOMM	Radio Frequency Communication
TCS Binary	Telephony Control Specification Binary
PDU	Protocol Data Unit

Bibliografía

ACL	Asynchronous Connection-Less Link
CID	Channel Identifier
ISM	Industry, Scientific and Medical
MHz	Megahertzio
GFSK	Gaussian Frequency-Shift Keying
PSK	Phase-Shift Keying
SCO	Synchronous Connection-Oriented
HMM	Hidden Markov Model
DTW	Dynamic Time Warping
SO	Sistema Operativo
UART	Universal Asynchronous Receiver-Transmitter
VR	Voice Recognition
IP	Internet Protocol
HNPT	Hospital Nacional de Parapléjicos de Toledo

Bibliografía

[1] G. Barroso de María, *Dispositivo automático de apoyo para uso de herramientas requeridas de movimiento manual de pinzado (Tesis de Máster),* A. Jardon Huete, Ed., Leganés: Universidad Carlos III de Madrid, 2012.

[2] A. J. Huete, *Proyecto PRESSMATIC, II Convocatoria de proyectos inclusivos de Fundación Universia*, 2013.

[3] A. Rosado García, *Protocolo de comunicación Bluetooth y control por voz para PRESSMATIC (Trabajo de Fin de Grado),* Leganés: Universidad Carlos III de Madrid, 2014.

[4] A. Vega Gómez, *Programación en Android accesible para el control de PRESSMATIC (Trabajo de Fin de Grado),* Leganés: Universidad Carlos III de Madrid, 2014.

[5] R. Martín Lopez, *Programación e integración de interfaz de pantalla táctil accesible para PRESSMATIC (Trabajo de Fin de Grado),* Leganés: Universidad Carlos III de Madrid, 2014.

[6] «Invest in Skane,» [En línea]. Available: http://www.investinskane.com/story-about-bluetooth. [Último acceso: 19 9 2014].

[7] «Bluetooth SIG Official Site, History,» [En línea]. Available: http://www.bluetooth.com/Pages/History-of-Bluetooth.aspx. [Último acceso: 18 9 2014].

[8] «IEEE Xplore,» [En línea]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&tp=&arnumber=1016473&sortType%3Dasc_p_Sequence%26filter%3DAND%28p_IS_Number%3A21872%29. [Último acceso: 19 9 2014].

[9] «L2CAP Bluetooth Developer Portal,» [En línea]. Available: https://developer.bluetooth.org/TechnologyOverview/Pages/L2CAP.aspx. [Último acceso: 19 9 2014].

[10] D. Cevallos, M. Paredes y A. Pilicita, «Comunicación de Datos,» 14 11 2011. [En línea]. Available: http://comunicaciondedatos.wordpress.com/. [Último acceso: 19 9 2014].

[11] «Radio-Electronics.com Bluetooth radio interface, modulation and channels,» [En línea]. Available: http://www.radio-electronics.com/info/wireless/bluetooth/radio-interface-modulation.php. [Último acceso: 19 9 2014].

[12] «Bluehack Seguridad en Bluetooth,» [En línea]. Available: http://bluehack.elhacker.net/proyectos/bluesec/bluesec.html. [Último acceso: 19 9 2014].

Bibliografía

[13] «Electrónica Facil. Formato de Paquetes Bluetooth,» [En línea]. Available: http://www.electronicafacil.net/tutoriales/Formato-paquetes-Bluetooth.php. [Último acceso: 19 9 2014].

[14] «Arduino Build Process,» [En línea]. Available: http://arduino.cc/en/Hacking/BuildProcess. [Último acceso: 19 9 2014].

[15] «EFE: Futuro,» 30 7 2013. [En línea]. Available: http://www.efefuturo.com/noticia/ios-contra-android-ventajas-y-caracteristicas/. [Último acceso: 19 9 2014].

[16] «Android Developers Web. SDK,» [En línea]. Available: http://developer.android.com/sdk/index.html. [Último acceso: 19 9 2014].

[17] «Maggie: Futuro, autonomía y diversión,» [En línea]. Available: http://portal.uc3m.es/portal/page/portal/actualidad_cientifica/actualidad/reportajes/archivo_reportajes/Maggie_futuro_autonomia_diversion/. [Último acceso: 19 9 2014].

[18] «TechHive.Speech Recognition Through the Decades,» 2 11 2011. [En línea]. Available: http://www.techhive.com/article/243060/speech_recognition_through_the_decades_how_we_ended_up_with_siri.html. [Último acceso: 19 9 2014].

[19] «Speech Recognition,» [En línea]. Available: http://en.wikipedia.org/wiki/Speech_recognition. [Último acceso: 19 9 2014].

[20] «HTK Speech Recognition,» [En línea]. Available: http://htk.eng.cam.ac.uk/. [Último acceso: 19 9 2014].

[21] «Inkscape. Tutoriales,» [En línea]. Available: http://www.inkscape.org/es/aprende/tutoriales/. [Último acceso: 19 9 2014].

[22] «Android Developers Web,» [En línea]. Available: http://developer.android.com/reference/packages.html. [Último acceso: 19 9 2014].

[23] «Blueterm Source Code,» [En línea]. Available: http://pymasde.es/blueterm/. [Último acceso: 19 9 2014].

[24] J. M. Bishop, Java: fundamentos de programación, 2º ed., Madrid: Addison-Wesley Iberoamericana, 2000.

[25] «EasyVR Downloads,» [En línea]. Available: http://www.veear.eu/downloads/. [Último acceso: 19 9 2014].

[26] «Getting Started with Arduino Tutorial,» [En línea]. Available: http://arduino.cc/en/Guide/HomePage. [Último acceso: 19 9 2014].

[27] B. O. J. Francisco, J. L. De La Cruz Fernández, M. Torres Roldán y J. A. Gistas Peyrona, «Consejo General de la Ingeniería Técnica Industrial,» 10 2004. [En línea]. Available: http://www.tecnicaindustrial.es/tifrontal/a-1434-Comunicacion-inalambrica-

Bluetooth.aspx. [Último acceso: 19 9 2014].

[28] J. Tomás Gironés, El gran libro de Android, 3ª ed., Barcelona: Marcombo, S.A., 2012.

[29] J. Tomás Gironés, El gran libro de Android avanzado, 1ª ed., Barcelona: Marcombo, S.A., 2012.